SERNAC
Servicio Nacional del Consumidor

PROTOTIPO MODELO IA

Contenido

1. Introducción 3
1.1. Propósito 3
1.2. Alcance 3
1.3. Público Objetivo 3
1.4. Definiciones 4
2. Requerimientos del Sistema 5
2.1 Arquitectura servidor: 5
2.2 Requisitos software 6
3. Descripción General del Sistema 6
3.1. Descripción del Sistema 6
3.2. Arquitectura específica del RAG SERNAC 7
3.2.1. GenAI API 7
3.2.2. ChromaDB 8
3.2.3. MLflow 8
3.2.4. Ollama Server 8
3.2.5. Proceso de Vectorización 8
3.2.6. Modelo LLM (Llama3) 9
4. Diseño de Software GenAI API 10
4.1. Diseño de Componentes 10
4.1.1 API Flask 10
4.1.2 ChromaDB Client 11
4.1.3 MLflow Client 11
4.1.4 Modelo LLM 12
4.2. Secuencia operaciones 12
4.2.1. /resumir_reclamos: 12
4.2.2. /buscar_reclamos_prob: 13
4.2.3. /vectorizar_tabla: 13
5. Implementación GenAI 14
5.1 Estructura de Directorios: 14
6. Uso GenAI API 16
6.1 Endpoint /resumir_reclamos: 16
6.2 Endpoint /buscar_reclamos_prob: 17
6.3 Endpoint /vectorizar_tabla: 19
7. Resultados pruebas masivas 20
7.1 Resultados /resumir_reclamos 20
7.1.1 Input pruebas 20
7.1.2 Respuesta ejemplo pruebas 20
7.1.3 Resultados pruebas 22
7.1.4 Conclusiones 25

7.2 Resultados /buscar_reclamos_prob 27
7.2.1 Input pruebas 27
7.2.2 Respuesta ejemplo pruebas 28
8. Referencias 30
9. Anexos 30

1. Introducción

1.1. Propósito

El propósito de este documento es proporcionar una descripción detallada del sistema RAG (Retrieval-Augmented Generation) implementado en SERNAC, en su etapa de prototipo, desarrollada entre los días **19 de Junio** y **12 de Julio** de 2024. Este sistema combina la generación de lenguaje natural con la recuperación de documentos relevantes para mejorar la precisión y relevancia de las respuestas generadas en base a datos institucionales.

1.2. Alcance

Este documento abarca todas las funcionalidades del sistema RAG, la generación de respuestas a preguntas mediante un modelo de lenguaje, la integración con ChromaDB para la recuperación de documentos relevantes, y el uso de MLflow para el rastreo y la gestión de experimentos de aprendizaje automático. También se incluye la exposición de una API RESTful utilizando Flask para permitir la interacción con el sistema.

El documento detalla la arquitectura del sistema, los componentes individuales y su implementación, así como las estrategias de pruebas y resultados. No se cubren aspectos relacionados con el diseño de interfaz de usuario, ya que el sistema está orientado a ofrecer un backend para la generación de respuestas asincrónicas.

1.3. Público Objetivo

Este documento está dirigido a desarrolladores de software, ingenieros de datos, administradores de sistemas, y cualquier otro personal técnico involucrado en el desarrollo, mantenimiento y despliegue del sistema RAG. Los lectores deben tener un conocimiento básico de desarrollo de software, arquitectura de sistemas, y tecnologías específicas como Flask, MLflow, y ChromaDB. El documento está estructurado para proporcionar la información necesaria de manera clara y detallada, permitiendo a los equipos técnicos comprender y trabajar con el sistema de manera efectiva.

1.4. Definiciones

Para facilitar la comprensión del documento, a continuación se presentan las definiciones de los términos y acrónimos utilizados:

RAG (Retrieval-Augmented Generation): Técnica que combina la generación de lenguaje natural con la recuperación de documentos relevantes para mejorar la precisión y relevancia de las respuestas.

MLflow: Plataforma de código abierto para gestionar el ciclo de vida del aprendizaje automático, incluyendo el rastreo de experimentos, el despliegue de modelos y la gestión de modelos en producción. (MLflow, 2024)

ChromaDB: Base de datos diseñada para gestionar y recuperar documentos y vectores de embeddings de manera eficiente. (ChromaDB, 2024)

Flask: Microframework de desarrollo web en Python que se utiliza para construir aplicaciones web y servicios RESTful.(Flask, 2024)

Embedding: Representación vectorial de datos (como palabras o documentos) en un espacio de características que facilita su procesamiento por algoritmos de aprendizaje automático.

LLM (Large Language Model): Modelo de lenguaje, como llama3, utilizado para tareas de procesamiento de lenguaje natural.

2. Requerimientos del Sistema

El sistema presentado en este documento se encuentra alojado en su totalidad en el server xxx.xxx.xxx.xxx con características de hardware y software detallados a continuación.

2.1 Arquitectura servidor:

El servidor que aloja la granja de servicios asociados al sistema RAG está ubicado en xxx.xxx.xxx.xxx con las siguientes características:

Atributo	Valor
CPU Architecture	x86_64
CPU op-mode	32-bit, 64-bit
CPU(s)	16
Thread(s) per core	1
Core(s) per socket	1
Socket(s)	8
Model name	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
CPU MHz	2199
BogoMIPS	4399.99
L1d cache	256 KiB
L1i cache	256 KiB
L2 cache	2 MiB
L3 cache	240 MiB
RAM	8 GB
Swap	980 MB

2.2 Requisitos software

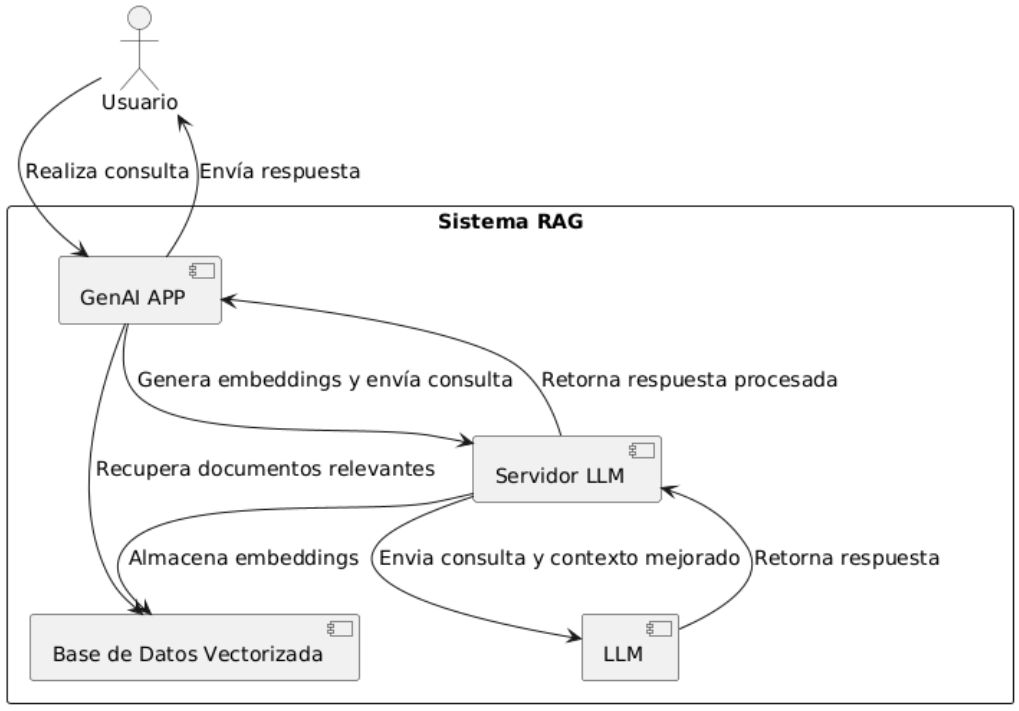
El software necesario mínimo para ejecutar la totalidad de servicios asociados es el siguiente:

1. Linux debian Debian 10.2.1-6
2. Ollama 0.1.32
3. Conda 23.3.1
4. Python 3.10
5. Creación ambiente DEV sobre Python 3.10 con las librerías contenidas en el anexo 1: requirements.txt

3. Descripción General del Sistema

3.1. Descripción del Sistema

El sistema RAG (Retrieval-Augmented Generation) está diseñado para mejorar la precisión y relevancia de las respuestas generadas mediante la combinación de un LLM con la recuperación de documentos relevantes. La arquitectura del sistema se divide en varias etapas clave que se pueden visualizar en la siguiente imagen:

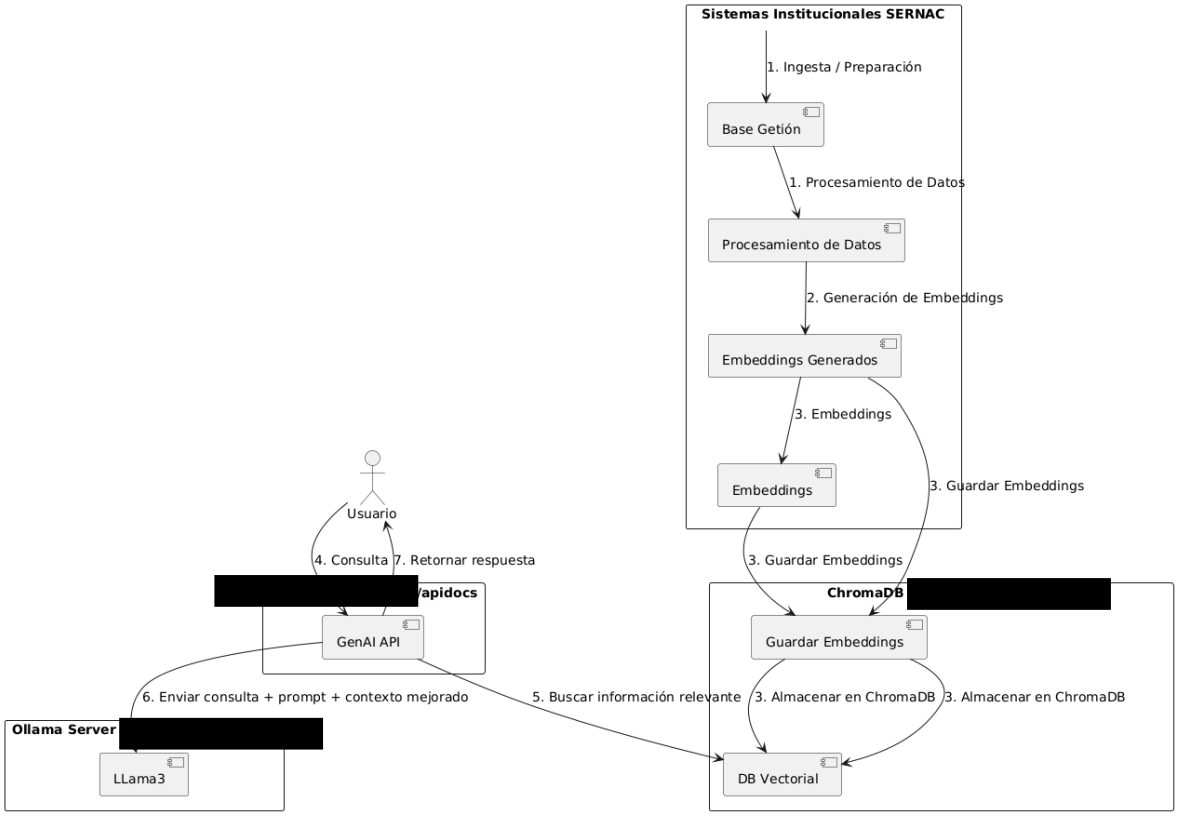


A grandes rasgos, los datos se procesan desde una base de datos estructurada para generar embeddings que se almacenan en una base de datos vectorial. Cuando un usuario realiza una

consulta, el sistema recupera la información relevante de la base de datos vectorial y utiliza un modelo de lenguaje para generar una respuesta enriquecida con el contexto adecuado. Esta respuesta se devuelve al usuario, proporcionando una experiencia más precisa y un análisis masivo de texto entre 4 y 10 minutos.

3.2. Arquitectura específica del RAG SERNAC

La arquitectura específica del sistema RAG implementado en la institución se puede dividir en diversos componentes, los que se pueden apreciar a continuación:



3.2.1. GenAI API

Descripción: La API Flask actúa como punto de entrada para las solicitudes del usuario. Maneja las solicitudes HTTP y coordina las operaciones entre los diferentes componentes del sistema.

Responsabilidades:

Recibir y procesar solicitudes HTTP: La API recibe las solicitudes del usuario y procesa los datos de entrada.

Validar y transformar datos: Asegura que los datos de entrada sean válidos y los transforma según sea necesario.

Coordinar la comunicación: Facilita la interacción entre el cliente de ChromaDB, el cliente de MLflow y el modelo LLM.

Enviar respuestas HTTP: Devuelve los resultados generados o recuperados en formato de respuesta HTTP.

3.2.2. ChromaDB

Descripción: El cliente de ChromaDB interactúa con la base de datos vectorial ChromaDB para almacenar y recuperar documentos utilizando embeddings vectoriales.

Responsabilidades:

Almacenar embeddings: Guarda las representaciones vectoriales de los datos.

Recuperar documentos relevantes: Busca y obtiene documentos relevantes en base a las consultas del usuario.

3.2.3. MLflow

Descripción: El cliente de MLflow se encarga de rastrear los experimentos, parámetros y métricas en MLflow.

Responsabilidades:

Iniciar y finalizar experimentos: Maneja la creación y finalización de experimentos en MLflow.

Registrar parámetros y métricas: Guarda los parámetros y métricas de los experimentos.

Guardar artefactos: Almacena los resultados generados durante los experimentos.

3.2.4. Ollama Server

Descripción: El Ollama Server genera embeddings a partir de los datos proporcionados.

Responsabilidades:

Procesar datos para generar embeddings: Convierte los datos en representaciones vectoriales.

Proveer embeddings generados: Facilita los embeddings generados al cliente de ChromaDB para su almacenamiento.

3.2.5. Proceso de Vectorización

Descripción: El proceso de vectorización transforma los datos en representaciones vectoriales que pueden ser almacenadas y consultadas eficientemente.

Responsabilidades:

Preparar datos: Prepara los datos para la generación de embeddings.

Almacenar embeddings: Guarda las representaciones vectoriales en ChromaDB.

3.2.6. Modelo LLM (Llama3)

Descripción: El modelo de lenguaje grande (LLM) genera respuestas basadas en el prompt y los documentos recuperados.

Responsabilidades:

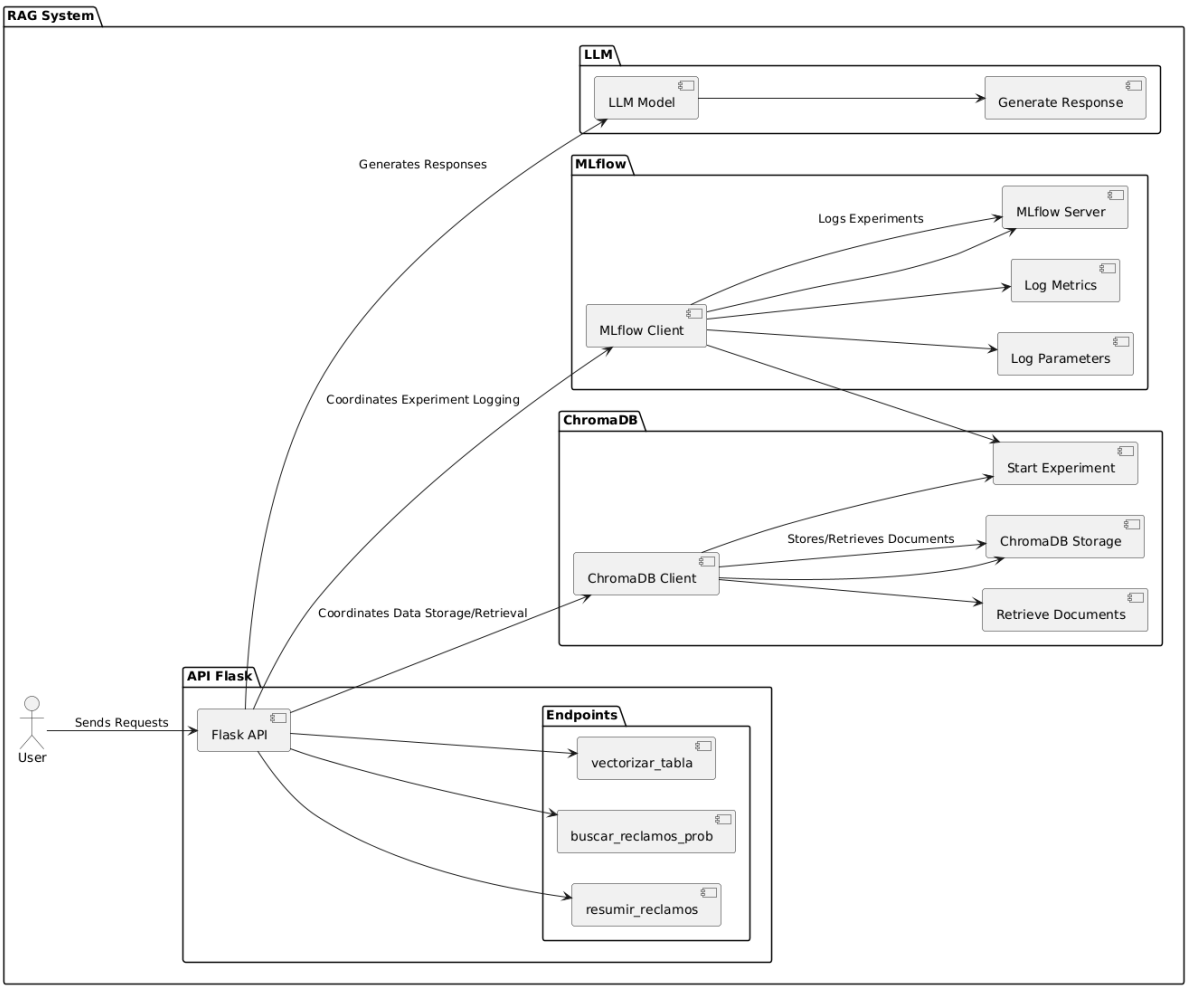
Recibir el prompt y el contexto: Procesa la consulta y el contexto proporcionado.

Generar respuestas adecuadas: Produce respuestas coherentes y contextualmente relevantes.

4. Diseño de Software GenAI API

4.1. Diseño de Componentes

Cada componente del sistema tiene un rol específico y se describe a continuación:



4.1.1 API Flask

Descripción: La API Flask maneja las solicitudes HTTP, coordina las operaciones entre los diferentes componentes y sirve como punto de entrada para los usuarios del sistema.

Responsabilidades:

Recibir y procesar solicitudes HTTP.

Validar y transformar los datos de entrada.

Coordinar la comunicación entre el cliente de ChromaDB, el cliente de MLflow y el modelo LLM.

Enviar respuestas HTTP con los datos generados o recuperados.

Endpoints Clave:

/resumir_reclamos: Endpoint para resumir reclamos desde el modelo.

/buscar_reclamos_prob: Endpoint para buscar reclamos problemáticos.

/vectorizar_tabla: Endpoint para vectorizar datos desde una tabla en PostgreSQL.

4.1.2 ChromaDB Client

Descripción: El cliente de ChromaDB interactúa con ChromaDB para almacenar y recuperar documentos utilizando embeddings vectoriales.

Responsabilidades:

Almacenar embeddings generados a partir de los datos del cliente.

Recuperar documentos relevantes de ChromaDB basados en consultas de los usuarios.

Operaciones Clave:

store_embeddings(): Función para almacenar embeddings en ChromaDB.

retrieve_documents(): Función para recuperar documentos relevantes de ChromaDB.

4.1.3 MLflow Client

Descripción: El cliente de MLflow se encarga de interactuar con MLflow para rastrear experimentos, parámetros y métricas.

Responsabilidades:

Iniciar y finalizar experimentos en MLflow.

Registrar parámetros, métricas y resultados de los experimentos.

Guardar artefactos generados durante los experimentos.

Operaciones Clave:

start_experiment(): Función para iniciar un nuevo experimento en MLflow.

log_parameters(): Función para registrar parámetros de los experimentos.

log_metrics(): Función para registrar métricas de los experimentos.

4.1.4 Modelo LLM

Descripción: El modelo de lenguaje grande (LLM) es responsable de generar respuestas basadas en el prompt y los documentos recuperados.

Responsabilidades:

Recibir el prompt y el contexto mejorado de la consulta.

Generar respuestas contextualmente adecuadas y coherentes.

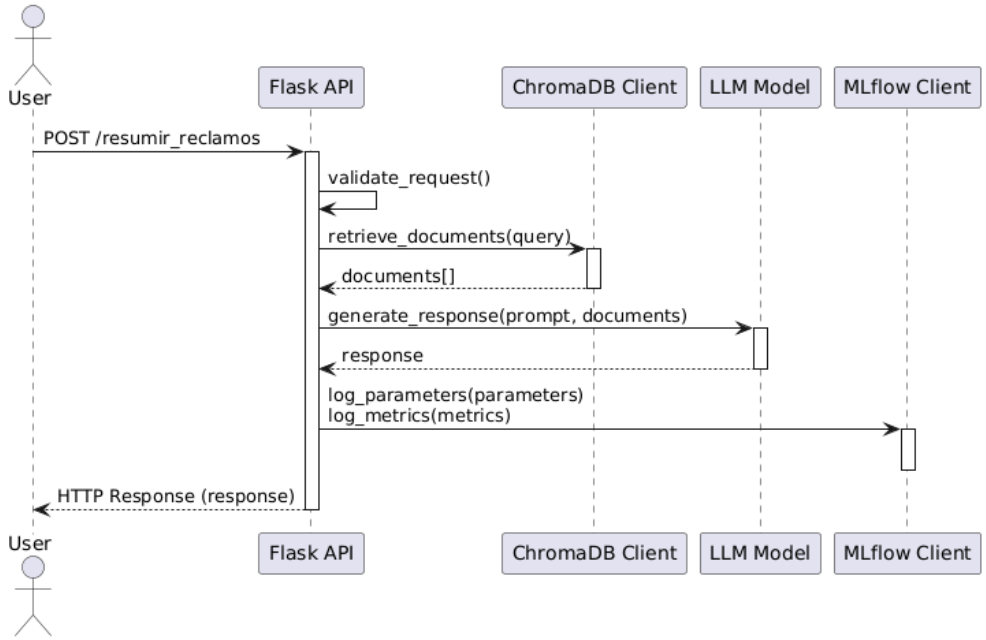
Operaciones Clave:

generate_response(): Función para generar respuestas utilizando el modelo LLM.

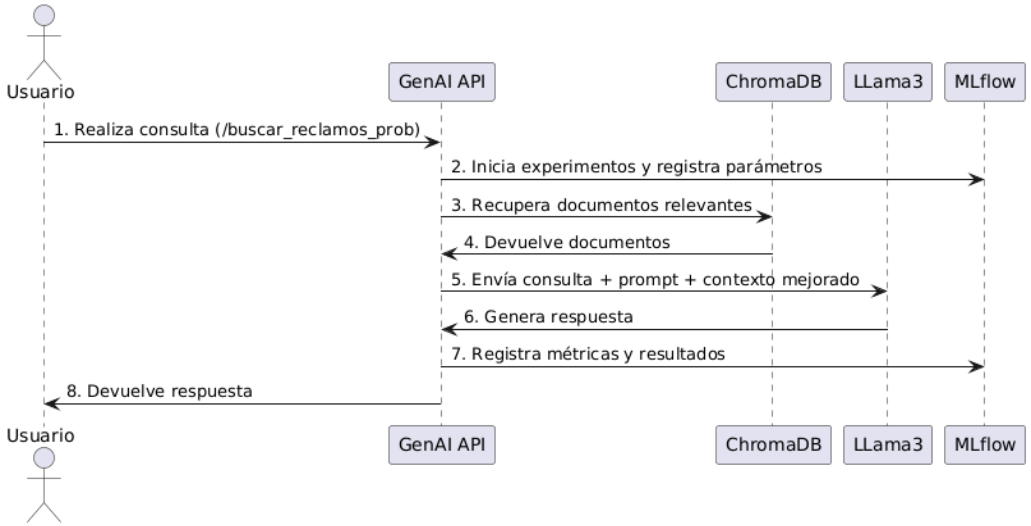
4.2. Secuencia operaciones

Los siguientes diagramas ilustran la secuencia de operaciones internas del sistema RAG, para cada uno de los endpoints de la API GenAI:

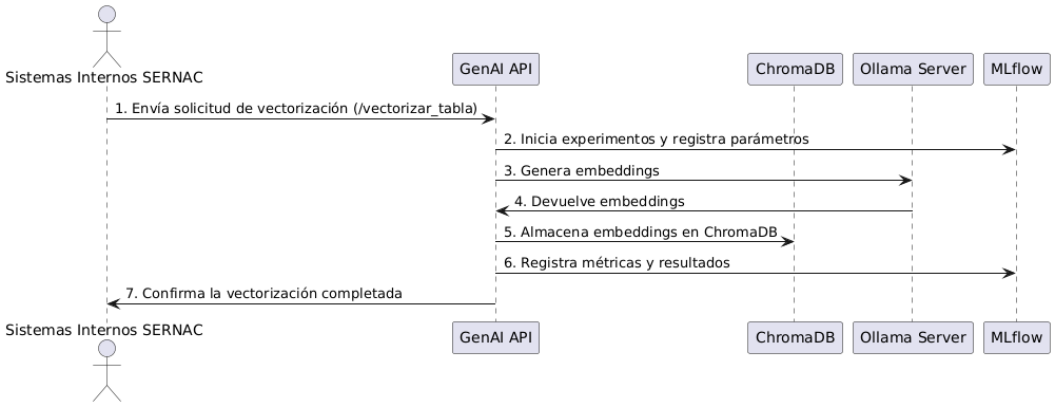
4.2.1. /resumir_reclamos:



4.2.2. /buscar_reclamos_prob:



4.2.3. /vectorizar_tabla:



5. Implementación GenAI

A continuación se describe el código fuente, la estructura de directorios y los archivos importantes del sistema, así como las funciones y métodos clave.

5.1 Estructura de Directorios:



5.2 Descripción de Archivos Importantes:

1-requirements.txt: Lista de dependencias del proyecto.

2-generate.yaml: Archivo de configuración de Swagger para la documentación de la API.

3-main.py: Archivo principal que inicia la aplicación Flask.

4-app/models.py:

CustomLLM: Clase que encapsula el modelo de lenguaje grande (LLM) y sus métodos.

generate(): Método para generar respuestas utilizando el modelo LLM.

5-app/routes.py:

resumir_reclamos(): Endpoint para resumir reclamos desde el modelo.

buscar_reclamos_prob(): Endpoint para buscar reclamos problemáticos.

vectorizar_tabla(): Endpoint para vectorizar datos desde una tabla en PostgreSQL.

generate_resume_response(): Función para manejar la generación de resúmenes.

generate_generic_response(): Función para manejar la generación de respuestas genéricas.

6-app/services.py:

generate_response(): Función para generar respuestas utilizando el modelo LLM, interactuar con ChromaDB y registrar métricas en MLflow.

vectorize_and_store_reclamos(): Función para vectorizar y almacenar reclamos en ChromaDB.

generate_resume(): Función específica para generar resúmenes de reclamos.

7-app/utils.py:

calculate_similarity(): Función para calcular la similitud entre textos.
calculate_bleu(): Función para calcular la métrica BLEU.
calculate_rouge(): Función para calcular las métricas ROUGE.
calculate_meteor(): Función para calcular la métrica METEOR.
calculate_accuracy(): Función para calcular la precisión.
calculate_metrics(): Función para calcular todas las métricas de evaluación de respuestas.

6. Uso GenAI API

A continuación se documentan los endpoints de la API, incluyendo la descripción del endpoint, el método HTTP, los parámetros de entrada y salida, y ejemplos de solicitud y respuesta.

6.1 Endpoint /resumir_reclamos:

Descripción: Genera un resumen de los reclamos desde un modelo de lenguaje basado en los parámetros proporcionados. Además, calcula diversas métricas para evaluar la calidad del resumen generado.

Método HTTP: POST

Parámetros de Entrada:

prompt (string): El prompt para generar una respuesta.
ground_truth (string): El texto de referencia para evaluar las métricas.
temperature (number): Temperatura de muestreo.
top_k (number): Parámetro de muestreo top-k.
top_p (number): Parámetro de muestreo top-p (nucleus sampling).
repetition_penalty (number): Penalización por repetición.
frequency_penalty (number): Penalización por frecuencia.
presence_penalty (number): Penalización por presencia.
run_name (string): El nombre de la ejecución en MLflow.
collection (string): El nombre de la colección en ChromaDB.

Ejemplo de solicitud:

```
curl -X POST http://localhost:5000/resumir_reclamos \ -H "Content-Type: application/json" \ -d '{ "prompt":
"Haz un resumen de las problemáticas de consumo contenidas en los reclamos",
"ground_truth": "No existe problemáticas "
, "temperature": 0.7
, "top_k": 50
, "top_p": 0.9
, "repetition_penalty": 1.0
, "frequency_penalty": 0.0
, "presence_penalty": 0.0
, "run_name": "Test_API"
, "collection": "reclamos"
}'
```

6.2 Endpoint /buscar_reclamos_prob:

Descripción: Genera una búsqueda de reclamos problemáticos desde un modelo de lenguaje basado en los parámetros proporcionados. Además, calcula diversas métricas para evaluar la calidad de la búsqueda generada.

Método HTTP: POST

Parámetros de Entrada:

prompt (string): El prompt para generar una respuesta.
ground_truth (string): El texto de referencia para evaluar las métricas.
temperature (number): Temperatura de muestreo.
top_k (number): Parámetro de muestreo top-k.
top_p (number): Parámetro de muestreo top-p (nucleus sampling).
repetition_penalty (number): Penalización por repetición.
frequency_penalty (number): Penalización por frecuencia.
presence_penalty (number): Penalización por presencia.
run_name (string): El nombre de la ejecución en MLflow.
collection (string): El nombre de la colección en ChromaDB.

Ejemplo de solicitud:

```
curl -X POST http://localhost:5000/buscar_reclamos_prob\
    -H "Content-Type: application/json" \
    -d '{
        "prompt": "Analiza reclamos que hablen de tarjetas a las que se ha solicitado su bloqueo permanente pero el banco  sigue cobrando comisión de mantención, administración",
        "ground_truth": "No existe reclamos asociados a la problemática descrita",
        "temperature": 0.2,
        "top_k": 50,
        "top_p": 0.9,
        "repetition_penalty": 1.0,
        "frequency_penalty": 0.0,
        "presence_penalty": 0.0,
        "run_name": "Test_API_buscar",
        "collection": "reclamos_financiero"
      }'
```

Ejemplo respuesta:

A continuación, se presenta un análisis de los casos que cumplen con el criterio de reclamos que hablan de tarjetas a las que se ha solicitado su bloqueo permanente pero el banco sigue cobrando comisión de mantención, administración.

En general, se observa que en estos casos, los consumidores han solicitado el cierre o bloqueo permanente de sus tarjetas de crédito, pero el banco sigue cobrando comisiones de mantención, administración o otros

conceptos. Esto sugiere que el banco no está cumpliendo con la solicitud del consumidor de cerrar o bloquear la tarjeta de manera efectiva.

En algunos casos, se ha observado que el banco ha cobrado comisiones de mantención después de que el consumidor ha cerrado la cuenta o ha solicitado el cierre. Esto puede ser considerado como una práctica comercial no transparente y que no respeta la voluntad del consumidor. En otros casos, se ha observado que el banco sigue enviando estados de cuenta a los consumidores a pesar de que ya no hay ninguna deuda vigente. Esto puede ser considerado como una forma de mantener la tarjeta activa y seguir cobrando comisiones, a pesar de que el consumidor ya no tiene interés en utilizarla. En general, se puede concluir que estos casos reflejan una problemática de consumo que se caracteriza por la falta de transparencia y respeto por la voluntad del consumidor. Los consumidores deben ser informados claramente sobre las condiciones de cierre o bloqueo de sus tarjetas de crédito y no deben ser obligados a seguir pagando comisiones de mantención o administración después de que han solicitado el cierre.

Listado de casos con mayor similitud a la problemática:
ID Reclamo: R2024W8888888
ID Reclamo: R2023W77777777

6.3 Endpoint /vectorizar_tabla:

Descripción: Vectoriza datos desde una tabla en PostgreSQL y almacena los vectores en ChromaDB.
Método HTTP: POST
Parámetros de Entrada:
table_name (string): Nombre de la tabla en PostgreSQL.
metadata_fields (array): Lista de campos para guardar como metadata.
text_field (string): Campo para guardar como texto a vectorizar.
batch_size (number): Tamaño del batch para procesar los datos.

Ejemplo de solicitud

```
curl -X POST http://localhost:5000/vectorizar_tabla\
    -H "Content-Type: application/json" \
    -d  '{
  "table_name": "nombre_de_la_tabla",
  "metadata_fields": ["CASO_NUMERO", "CASO_FECHA_CREACION", "PROVEEDOR_RUT"],
  "text_field": "RECLAMO_DESCRIPCION",
  "batch_size": 100
}'
```

7. Resultados pruebas masivas

7.1 Resultados /resumir_reclamos

Para una misma pregunta y ground truth se efectuó ejecuciones masivas de GenAI API con distintas configuraciones de parámetros para analizar sus métricas de rendimiento de la siguiente manera:

```
# Rango de parámetros
temperature_values = [round(i * 0.1, 1) for i in range(11)]  # De 0.0 a 1.0 con paso 0.1
top_p_values = [round(0.7 + i * 0.05, 2) for i in range(6)]  # De 0.7 a 0.95 con paso 0.05

# Generar respuestas para cada combinación de parámetros
for temp in temperature_values:
    for top_p in top_p_values:
        try:
            print(f"Running with temperature={temp} and top_p={top_p}")
            question_response = generate_response(
                llm, header_prompt, question_prompt, collection, ground_truth,
                temperature=temp, top_k=50, top_p=top_p,
                repetition_penalty=1.0, frequency_penalty=0.0, presence_penalty=0.0
            )
            print(f"Response for temperature={temp} and top_p={top_p}:\n{question_response}\n")
        except Exception as e:
            print(f"Error with temperature={temp} and top_p={top_p}: {e}")
```

7.1.1 Input pruebas

El input utilizado para las pruebas fue el siguiente:

question_prompt = (

'Haz un resumen de los reclamos analizados el cual debe titularse Análisis Diaro de Reclamos. ' +

'Nunca menciones un reclamo en particular, ni su número ni id.' +

'Muestra las problemáticas de consumo desde las más recurrentes a las menos recurrentes, indicando su porcentaje de ocurrencia. '+

'Identifica entre 6 y 10 problemáticas de consumo. ' +

'Finalmente, analiza y haz una conclusión de esto, de tipo ejecutivo, dirigido a un perfil de abogado que necesita obtener conclusiones.'

)

7.1.2 Respuesta ejemplo pruebas

Una ejemplo de respuesta tipo, para temperatura 0.0 y top p 0.7 es la siguiente:

Análisis Diario de Reclamos

En el análisis diario de reclamos, se han identificado las siguientes problemáticas de consumo, clasificadas por su frecuencia de ocurrencia:

*1. **Problemas con la calidad del producto o servicio** (35%): La mayoría de los reclamos se refieren a la insatisfacción con la calidad del producto o servicio ofrecido, lo que puede ser debido a defectos, errores o incompatibilidades.*

*2. **Dificultades en el proceso de devolución o reembolso** (20%): Los clientes han reportado problemas al intentar devolver o reembolsar productos o servicios, lo que puede generar frustración y pérdida de confianza en la marca.*

*3. **Incongruencias en la información publicitaria** (15%): Los reclamos también se refieren a la falta de claridad o precisión en la información publicitaria, lo que puede generar expectativas no cumplidas y desilusión.*

*4. **Problemas con la atención al cliente** (10%): Los clientes han reportado problemas al intentar comunicarse con la empresa, como la falta de respuesta o la ineficacia en la resolución de problemas.*

*5. **Incongruencias en la entrega o envío** (5%): Los reclamos también se refieren a problemas con la entrega o envío de productos, como la demora o la pérdida de paquetes.*

*6. **Problemas con la facturación o cobro** (5%): Los clientes han reportado problemas con la facturación o cobro, como la falta de claridad en los términos y condiciones o la ineficacia en la resolución de problemas de cobro.*

*7. **Problemas con la garantía o protección** (3%): Los reclamos también se refieren a problemas con la garantía o protección ofrecida, como la falta de claridad en los términos o la ineficacia en la resolución de problemas.*

*8. **Problemas con la comunicación** (2%): Los clientes han reportado problemas con la comunicación, como la falta de claridad o la ineficacia en la resolución de problemas.*

*9. **Problemas con la seguridad y privacidad** (1%): Los reclamos también se refieren a problemas con la seguridad y privacidad, como la falta de protección de datos personales o la ineficacia en la resolución de problemas.*

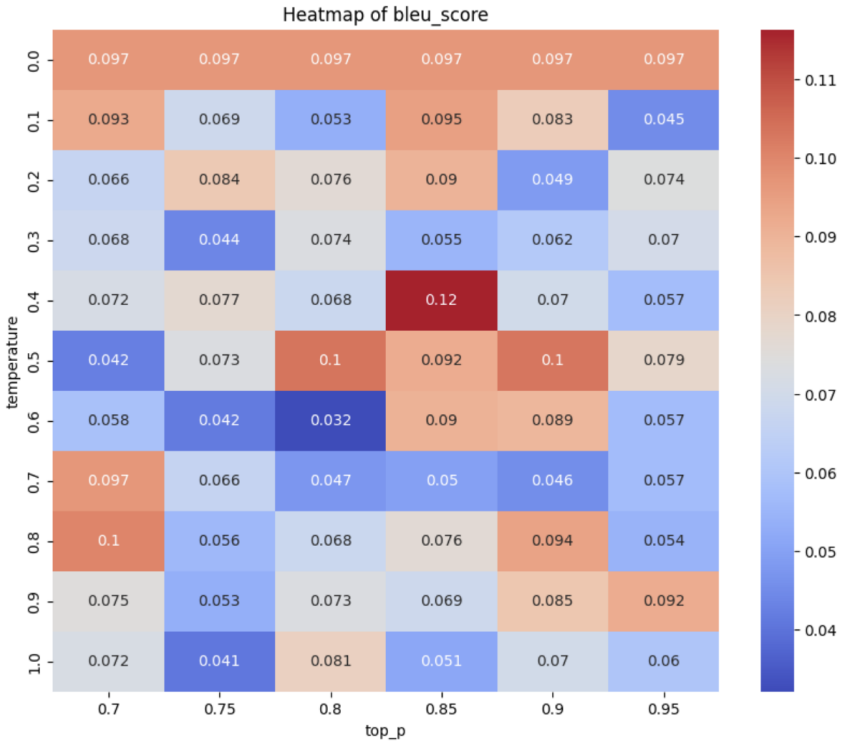
*10. **Otros problemas** (1%): Los reclamos también se refieren a otros problemas no clasificados en las categorías anteriores, como la falta de flexibilidad en las políticas de devolución o reembolso.*

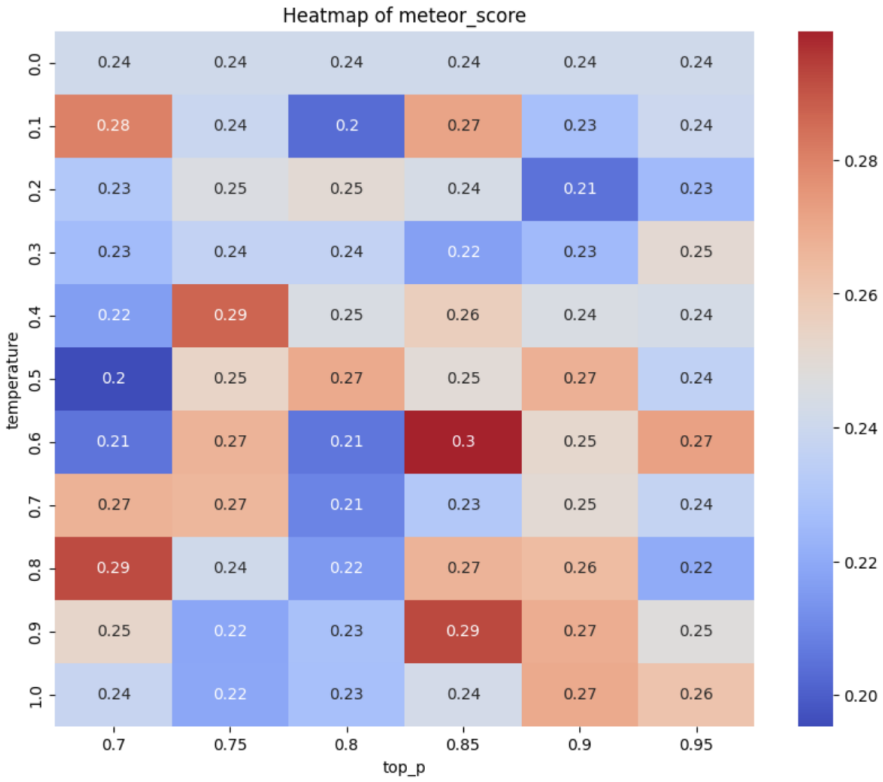
Conclusión

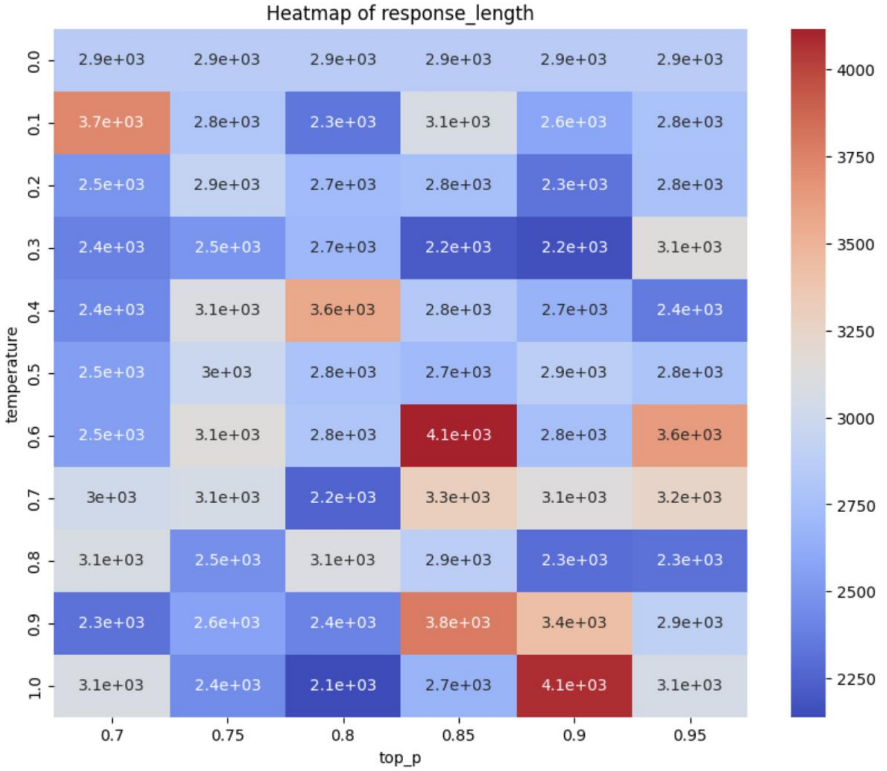
En general, los reclamos se refieren a problemas de consumo que pueden ser resueltos con un enfoque en la calidad del producto o servicio, la claridad en la información publicitaria y la eficacia en la atención al cliente. Es importante que las empresas tengan un enfoque en la resolución de problemas y la satisfacción del cliente para evitar la pérdida de confianza y la desilusión. Es fundamental que las empresas tengan un sistema de seguimiento y resolución de reclamos efectivo para garantizar la satisfacción del cliente y la protección de la marca.

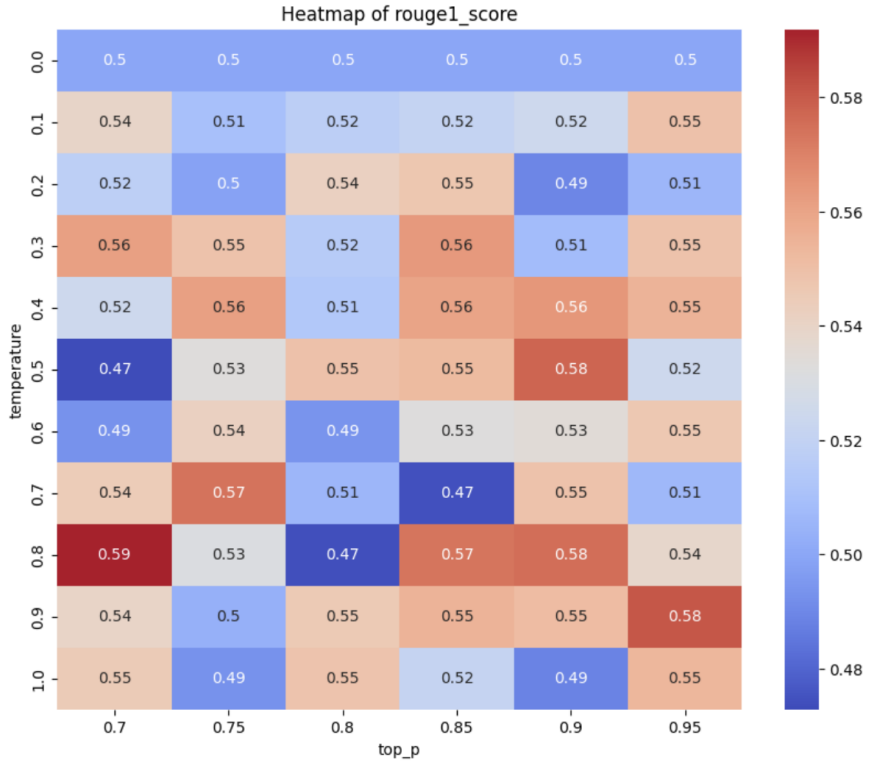
7.1.3 Resultados pruebas

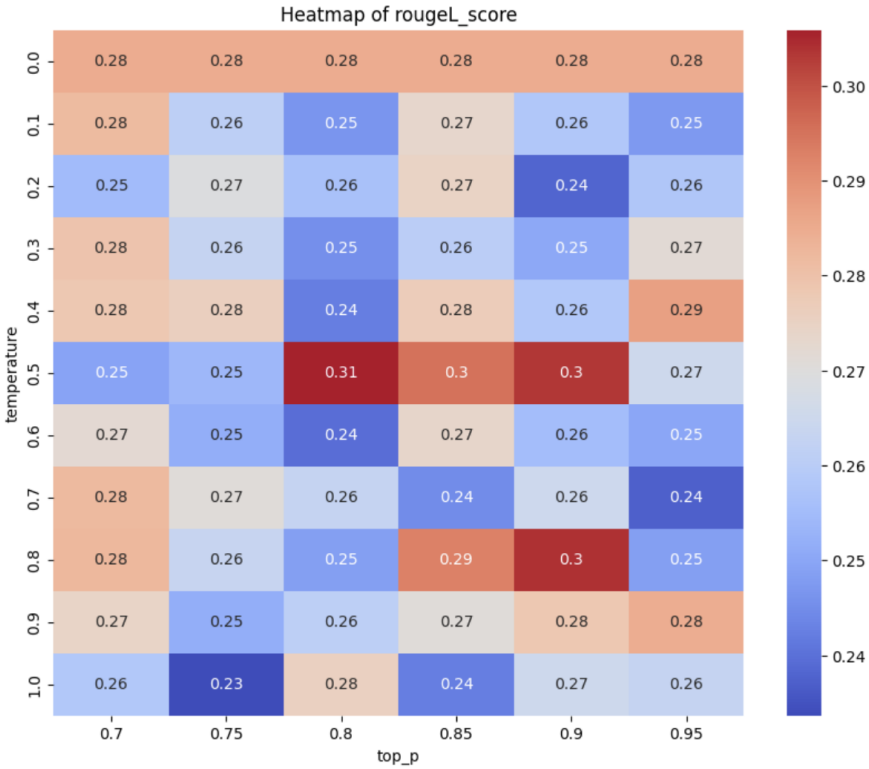
Los resultados a las pruebas masivas fueron los siguientes, donde se analiza el valor de distintas métricas como bleu score, meteor score y otros, en base a combinación de variables de entrada como temperatura y top p:

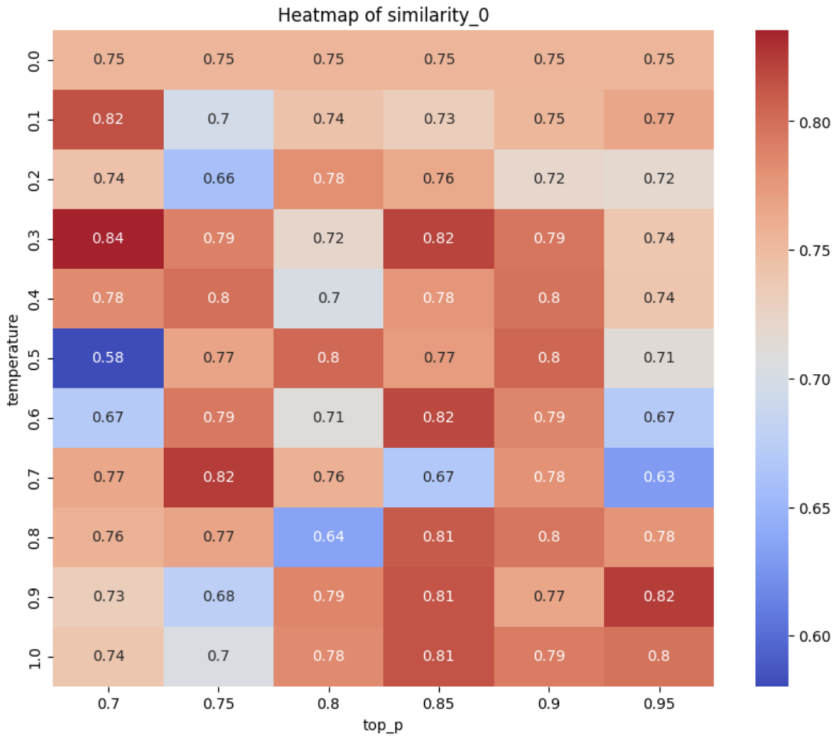



Heatmap of meteor_score
temperature
top_p
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.7 0.75 0.8 0.85 0.9 0.95
0.24 0.24 0.24 0.24 0.24 0.24
0.28 0.24 0.2 0.27 0.23 0.24
0.23 0.25 0.25 0.24 0.21 0.23
0.23 0.24 0.24 0.22 0.23 0.25
0.22 0.29 0.25 0.26 0.24 0.24
0.2 0.25 0.27 0.25 0.27 0.24
0.21 0.27 0.21 0.3 0.25 0.27
0.27 0.27 0.21 0.23 0.25 0.24
0.29 0.24 0.22 0.27 0.26 0.22
0.25 0.22 0.23 0.29 0.27 0.25
0.24 0.22 0.23 0.24 0.27 0.26
0.28 0.26 0.24 0.22 0.20


Heatmap of response_length
temperature
top_p
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.7 0.75 0.8 0.85 0.9 0.95
2.9e+03 2.9e+03 2.9e+03 2.9e+03 2.9e+03 2.9e+03
3.7e+03 2.8e+03 2.3e+03 3.1e+03 2.6e+03 2.8e+03
2.5e+03 2.9e+03 2.7e+03 2.8e+03 2.3e+03 2.8e+03
2.4e+03 2.5e+03 2.7e+03 2.2e+03 2.2e+03 3.1e+03
2.4e+03 3.1e+03 3.6e+03 2.8e+03 2.7e+03 2.4e+03
2.5e+03 3e+03 2.8e+03 2.7e+03 2.9e+03 2.8e+03
2.5e+03 3.1e+03 2.8e+03 4.1e+03 2.8e+03 3.6e+03
3e+03 3.1e+03 2.2e+03 3.3e+03 3.1e+03 3.2e+03
3.1e+03 2.5e+03 3.1e+03 2.9e+03 2.3e+03 2.3e+03
2.3e+03 2.6e+03 2.4e+03 3.8e+03 3.4e+03 2.9e+03
3.1e+03 2.4e+03 2.1e+03 2.7e+03 4.1e+03 3.1e+03
4000 3750 3500 3250 3000 2750 2500 2250


Heatmap of rouge1_score
temperature
top_p
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.7 0.75 0.8 0.85 0.9 0.95
0.5 0.5 0.5 0.5 0.5 0.5
0.54 0.51 0.52 0.52 0.52 0.55
0.52 0.5 0.54 0.55 0.49 0.51
0.56 0.55 0.52 0.56 0.51 0.55
0.52 0.56 0.51 0.56 0.56 0.55
0.47 0.53 0.55 0.55 0.58 0.52
0.49 0.54 0.49 0.53 0.53 0.55
0.54 0.57 0.51 0.47 0.55 0.51
0.59 0.53 0.47 0.57 0.58 0.54
0.54 0.5 0.55 0.55 0.55 0.58
0.55 0.49 0.55 0.52 0.49 0.55
0.58 0.56 0.54 0.52 0.50 0.48


Heatmap of rougeL_score
temperature
top_p
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
0.7 0.75 0.8 0.85 0.9 0.95
0.28 0.28 0.28 0.28 0.28 0.28
0.28 0.26 0.25 0.27 0.26 0.25
0.25 0.27 0.26 0.27 0.24 0.26
0.28 0.26 0.25 0.26 0.25 0.27
0.28 0.28 0.24 0.28 0.26 0.29
0.25 0.25 0.31 0.3 0.3 0.27
0.27 0.25 0.24 0.27 0.26 0.25
0.28 0.27 0.26 0.24 0.26 0.24
0.28 0.26 0.25 0.29 0.3 0.25
0.27 0.25 0.26 0.27 0.28 0.28
0.26 0.23 0.28 0.24 0.27 0.26
0.30 0.29 0.28 0.27 0.26 0.25 0.24



7.1.4 Conclusiones

A raíz de lo anterior se puede concluir lo siguiente:

BLEU Score:

Las configuraciones con temperature en 0.5 y top_p entre 0.85 y 0.9 tienden a producir los valores más altos de BLEU Score. Esto sugiere que una temperatura moderada combinada con una probabilidad de muestreo ligeramente ajustada favorece la precisión de la generación de texto.

Los valores más bajos se observan con temperaturas extremas (0.0 y 1.0), lo que indica que tanto la aleatoriedad excesiva como la falta de aleatoriedad pueden perjudicar la calidad de las respuestas generadas.

Meteor Score:

Similar al BLEU Score, las configuraciones con temperature en 0.6 y top_p entre 0.8 y 0.85 presentan los puntajes más altos. Esto respalda la conclusión de que una temperatura moderada y una probabilidad de muestreo ajustada proporcionan mejores resultados.

Las temperaturas extremas nuevamente muestran un rendimiento inferior.

Similarity Score:

El score de similitud más alto se observa con temperature en 0.5 y top_p en 0.85. Esto refuerza la idea de que estas configuraciones optimizan la calidad de las respuestas generadas.

Configuración Óptima:

Basado en la mayoría de las métricas (BLEU Score, Meteor Score, Similarity Score), la configuración óptima parece ser temperature en 0.5 y top_p en 0.85 a 0.9.

Esta configuración equilibra bien la calidad de las respuestas generadas y el tiempo de respuesta, lo que es crucial para la eficiencia del sistema.

Impacto de la Temperatura:

Temperaturas extremas (0.0 y 1.0) no son recomendables ya que perjudican tanto la calidad de las respuestas como la eficiencia del sistema.

Una temperatura moderada (0.5 a 0.7) es ideal para mantener un equilibrio entre diversidad y coherencia en las respuestas generadas.

Impacto de top_p:

Valores de top_p entre 0.85 y 0.9 son preferibles ya que permiten una selección de tokens más precisa, mejorando la calidad de las respuestas generadas.

7.2 Resultados /buscar_reclamos_prob

Como se ha mencionado anteriormente, la búsqueda de reclamos dada una problemática específica busca generar un listado de los N° de casos de reclamos vectorizados que cumplen con un criterio especificado en el prompt de entrada. Para una misma pregunta y ground truth se efectuó ejecuciones masivas de GenAI API con distintas configuraciones de parámetros para analizar sus métricas de rendimiento de la siguiente manera:

```
url = "http://localhost:5000/buscar_reclamos_prob"
headers = {"Content-Type": "application/json"}

# Rango de valores para temperature y top_p
temperature_values = [round(i * 0.1, 1) for i in range(11)]  # [0.0, 0.1, ..., 1.0]
top_p_values = [round(0.5 + i * 0.05, 2) for i in range(11)]  # [0.5, 0.55, ..., 1.0]

for temperature in temperature_values:
    for top_p in top_p_values:
        payload = {
            "prompt": "Analiza reclamos que hablen de tarjetas a las que se ha solicitado su bloqueo permanente pero el banco sigue cobrando comisión de
            "ground_truth": " ",
            "temperature": temperature,
            "top_k": 50,
            "top_p": top_p,
            "repetition_penalty": 1.0,
            "frequency_penalty": 0.0,
            "presence_penalty": 0.0,
            "run_name": "Test_API",
            "collection": "reclamos_financiero"
        }

        response = requests.post(url, headers=headers, data=json.dumps(payload))

        # Manejo de la respuesta
        if response.status_code == 200:
            print(f"Success: Temperature={temperature}, Top_p={top_p}")
            print(response.json())
        else:
            print(f"Failed: Temperature={temperature}, Top_p={top_p}")
            print(response.text)
```

7.2.1 Input pruebas

Para las pruebas masivas se hizo la búsqueda con el siguiente prompt:

Analiza reclamos que hablen de tarjetas a las que se ha solicitado su bloqueo permanente pero el banco sigue cobrando comisión de mantención, administración.
Aquí te dejo algunos casos de ejemplo que cumplen ese criterio:

Ejemplo 1- Tengo 2 tarjetas de crédito en el banco Junio 2023 , lo cual el banco me cerro las cuentas, fui al banco para cerrar personalmente el cierre, y al mes siguiente me estaban cobrando la mantención ya que había quedado algo pendiente y al otro mes volvieron a cobrar la mantención ya que hace 5 meses de cerro la cuenta.

Ejemplo 2- En noviembre del 2023 solicité cerrar todos mis productos Itau (Cuenta corriente, Línea de Crédito, Tarjeta de Crédito) Hasta el día de hoy sigo recibiendo estados de cuenta de la tarjeta de crédito a pesar de que ya no hay ninguna deuda vigente Tengo correos de mi ex ejecutiva de cuenta en los que me asegura que todo está cerrado. Ya en Enero tuve que pagar gastos de mantención de la tarjeta de crédito

Al igual que en el caso de resúmenes de reclamos se hizo pruebas masivas para distintas configuraciones de temperatura y top p.

7.2.2 Respuesta ejemplo pruebas

Una respuesta de ejemplo es la siguiente:

A continuación, se presenta un análisis de los casos que cumplen con el criterio de reclamos que hablan de tarjetas a las que se ha solicitado su bloqueo permanente pero el banco sigue cobrando comisión de mantención, administración.

En general, se observa que en estos casos, los consumidores han solicitado el cierre o bloqueo permanente de sus tarjetas de crédito, pero el banco sigue cobrando comisiones de mantención, administración o otros conceptos. Esto sugiere que el banco no está cumpliendo con la solicitud del consumidor de cerrar o bloquear la tarjeta de manera efectiva.

En algunos casos, se ha observado que el banco ha cobrado comisiones de mantención después de que el consumidor ha cerrado la cuenta o ha solicitado el cierre. Esto puede ser considerado como una práctica comercial no transparente y que no respeta la voluntad del consumidor. En otros casos, se ha observado que el banco sigue enviando estados de cuenta a los consumidores a pesar de que ya no hay ninguna deuda vigente. Esto puede ser considerado como una forma de mantener la tarjeta activa y seguir cobrando comisiones, a pesar de que el consumidor ya no tiene interés en utilizarla. En general, se puede concluir que estos casos reflejan una problemática de consumo que se caracteriza por la falta de transparencia y respeto por la voluntad del consumidor. Los consumidores deben ser informados claramente sobre las condiciones de cierre o bloqueo de sus tarjetas de crédito y no deben ser obligados a seguir pagando comisiones de mantención o administración después de que han solicitado el cierre.

Listado de casos con mayor similitud a la problemática:
ID Reclamo: R2024W888888888
ID Reclamo: R2023W999999999

Independiente de la configuración, en todos los casos muestra los mismos reclamos, cambiando de acuerdo a su temperatura el texto introductorio de ésta.

Este comportamiento se debe a que el listado de reclamos depende de la “búsqueda por similitud” que se realiza sobre la base de datos vectorizada ChromaDB. Cuando se realiza una búsqueda por similitud, ChromaDB compara el embedding del prompt con los embeddings almacenados utilizando una métrica de similitud, como la distancia coseno. Esta métrica mide el ángulo entre dos vectores en un espacio multidimensional, donde un ángulo más pequeño indica una mayor similitud.

Así, la búsqueda y recuperación de embeddings no pasa por la configuración de hyperparámetros del modelo.

8. Referencias

1. MLflow. (n.d.). *MLflow: An open source platform for the machine learning lifecycle*. Retrieved July 12, 2024 from https://mlflow.org/docs/latest/introduction/index.html
2. Chroma Core. (n.d.). *Chroma: The open source embedding database*. Retrieved July 12, 2024 from https://github.com/chroma-core/chroma
3. Pallets Projects. (n.d.). *Flask: A lightweight WSGI web application framework*. Retrieved July 12, 2024 from https://flask.palletsprojects.com/en/3.0.x/
4. Ollama. (n.d.). *Ollama GitHub repository*. GitHub. Retrieved July 14, 2024, from https://github.com/ollama/ollama
5. SERNAC. (Julio, 2024). RAG API Source Code. GitLab. Retrieved July 14, 2024, from https://git.sernac.cl/ugdin/rag_api.rag

9. Anexos