Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

LLMs y RAG para una mejor interacción con JIRA

Estudiante: Aróstegui García, Alberto

Directores: Egaña Aranguren, Mikel; López Novoa, Unai

Curso: 2023-2024

Fecha: 22 de mayo de 2024

Resumen:
Este trabajo se enfoca en explorar los límites de los ensayos a tracción para probetas de materiales compuestos.

Palabras Clave: *Materiales Compuestos*

Abstract:
English

Key Words:

Laburpena:
Euskera

Gako-hitzak:

Índice

Abreviaturas **8**

1 Contexto **9**
1.1 Gestión de proyectos 9
1.2 Herramientas 9
1.2.1 JIRA 9
1.2.2 Git - Gitlab 10
1.3 JiraGPT Next 10

2 Planificación **12**
2.1 Tareas 12
2.1.1 Gestión y planificación 12
2.1.2 Investigación 13
2.1.3 Ideas sistemas RAG 15
2.1.4 Definición de pruebas 15
2.1.5 Costes de software 16
2.1.6 Costes de mano de obra 16
2.1.7 Costes de hardware 16

3 Tecnologías **17**
3.1 Modelos grandes de lenguaje 17
3.2 Retrieval Augmented Generation 17
3.2.1 Funcionamiento 18
3.3 Word embeddings 18
3.4 Ontologías 19
3.5 Grafos de conocimiento 19

4 Diseño **21**
4.1 Estado inicial 21
4.2 Propuestas 21
4.2.1 Ontología 22
4.2.2 Embeddings 23
4.2.3 Grafos de conocimiento 24

5 Implementación **25**
5.1 Implementación de RAG: Embeddings 25
5.1.1 Extracción de datos 25
5.1.2 Interacción con el modelo 25
5.2 Implementación de RAG: Ontología 26

5.2.1 Interacción con el modelo 26

Referencias **28**

Índice de figuras

1 Esquema de división del trabajo 12
2 Esquema de funcionamiento de una arquitectura RAG. 18
3 Diagrama de RAG con ontología 22
4 Diagrama de RAG con embeddings 24

Índice de tablas

1 Reuniones 12
2 Investigación sobre el funcionamiento de los LLMs 13
3 Langchain y abstracción de LLMs 13
4 Técnicas de RAG 14
5 Ontologías, knowledge graphs y embeddings 14
6 Pipelines de RAG 14
7 Creación de nuevo Dataset 15
8 Modificación del benchmark 15

Abreviaturas

API Application Programming Interface

JQL Jira Query Language

LLM Large Language Model

RAG Retrieval Augmented Generation

PLN Procesamiento del Lenguaje Natural

1. Contexto

Este trabajo de fin de grado responde a las necesidades de la empresa LKS Next-GobTech, una empresa de desarrollo de software con enfoque en la innovación. De cara a comprender los motivos por los que esta empresa requiere de lo estudiado en este trabajo, se han de poner en contexto las herramientas y metodologías que utilizan para mejorar la calidad del producto que desarrollan. Partiendo del TFG de un compañero de escuela, Joel García Escribano [1], que consiste en un asistente conversacional que genera consultas JQL a partir de preguntas hechas con lenguaje natural, se ha estudiado la posibilidad de añadir una arquitectura de RAG (Retieval Augmented Generation) para aumentar la precisión de las respuestas que ofrece.

A continuación, se detallan las herramientas y metodologías que utiliza LKS Next-GobTech para la gestión de proyectos y se introduce la herramienta JiraGPTNext, que es el objeto de estudio de este trabajo.

1.1. Gestión de proyectos

La gestión de proyectos es el conjunto de metodologías utilizadas para coordinar la organización, la motivación y el control de recursos con el fin de alcanzar un objetivo. En el caso del desarrollo de software, ha sido un tópico controversial y de relevancia durante su historia, según la conocida ley de Brooks, expuesta en su libro *The mythical man-month* [2], añadir desarrolladores a un proyecto que va detrás del plazo solo hará que se retrase. Dentro de LKS Next-GobTech, donde se coordinan varios proyectos a la vez, resulta crucial una buena organización y división del trabajo en grupos preestablecidos al inicio de estos, con el fin de llevar un óptimo desarrollo en el que se cumplan los plazos establecidos.

En vistas de lo expuesto, parece obvia la necesidad de una herramienta de software capaz de suplir las necesidades ímplicitas en el desarrollo de software, por lo que dentro de esta empresa, se utiliza una herramienta de software llamada *JIRA*.

1.2. Herramientas

1.2.1. JIRA

JIRA es una herramienta de software propietario desarrollada por Atlassian para coordinar proyectos basados en tareas, llamadas incidencias dentro de la jerga de la aplicación. Esta herramienta sirve tanto para uso interno, como para que acceda el cliente, pudiendo encontrar un punto centralizado donde compartir información sobre el progreso y el estado del proyecto.

Las incidencias son la división atómica de paquetes de trabajo, que representan una tarea cuantificable asignable a un desarrollador y que ayudan a medir el desarrollo llevado a cabo. Al disponer de estados para las incidencias, se puede consultar de manera sencilla cómo progresa el proyecto.

Dentro de estas se pueden registrar distintos datos, como el tiempo que se prevé que va a tomar la tarea y el tiempo real que toma, mediante registros de trabajo, medidos en horas. Asimismo, se puede incluir información de interés para quien vaya a ser asignado el desarrollo de la incidencia, como una descripción, un resumen o enlaces externos a documentación relevante.

En un proyecto JIRA gestionado en LKS Next-GobTech se gestiona un flujo para las incidencias detallado a continuación: el desarrollador que la realice marcará la incidencia como hecha, a lo que un desarrollador senior validará el trabajo realizado y decidirá si es correcto o si ha de se mejorado. Una vez confirmado, se marcará como validada y podrá pasar a la vista del cliente, que podrá comprobar el trabajo realizado.

1.2.2. Git - Gitlab

Al igual que se necesita controlar el estado de trabajos en el proyecto, también es necesario llevar un control de versiones para un óptimo desarrollo de software. En el caso de LKS Next-GobTech se utiliza Git [3] como herramienta y Gitlab como punto centralizado donde guardar los repositorios.

Gitlab es una plataforma que permite gestionar las versiones del software y la colaboración entre desarrolladores. De esta manera, se crea un repositorio para cada proyecto que tiene la empresa y para cada uno de estos repositorios se otorgan permisos de modificación a los desarrolladores que vayan a trabajar en ese proyecto.

Además, se utiliza la integración de JIRA con Gitlab para relacionar las incidencias con cambios realizados en el repositorio asignado al proyecto, de manera que tanto la confirmación del trabajo realizado como del tiempo invertido pueden ser contrastados.

1.3. JiraGPT Next

Partiendo del trabajo realizado por Joel García, se dispone de JiraGPT Next como una herramienta que ayuda a recuperar incidencias filtradas utilizando lenguaje natural. De esta manera, una persona que no posea conocimiento técnico en la generación de consultas JQL podrá filtrar incidencias facilmente.

Tras esta herramienta se encuentra una llamada de API a un LLM que, utilizando

una plantilla para guiar al modelo, pedirá que se traduzca la pregunta en lenguaje natural a una consulta JQL que responda a lo que se pide.

La idea detrás de este nuevo trabajo es realizar un estudio de la mejora de precisión obtenida utilizando arquitecturas RAG. Para ello, se propone modificar la estructura que se sigue para la generación de consultas JQL utilizando LangChain y bases de conocimiento de las que recuperar información relevante para la generación de la consulta.

Con este estudio se pretende observar si las distintas arquitecturas propuestas suponen un cambio significativo en la precisión de las consultas generadas.

2. Planificación

En esta sección se detallará la planificación del trabajo de fin de grado, en la que se incluirán los objetivos, la metodología y el cronograma de trabajo. Además, se incluyen los recursos necesarios para llevar a cabo el proyecto.

2.1. Tareas

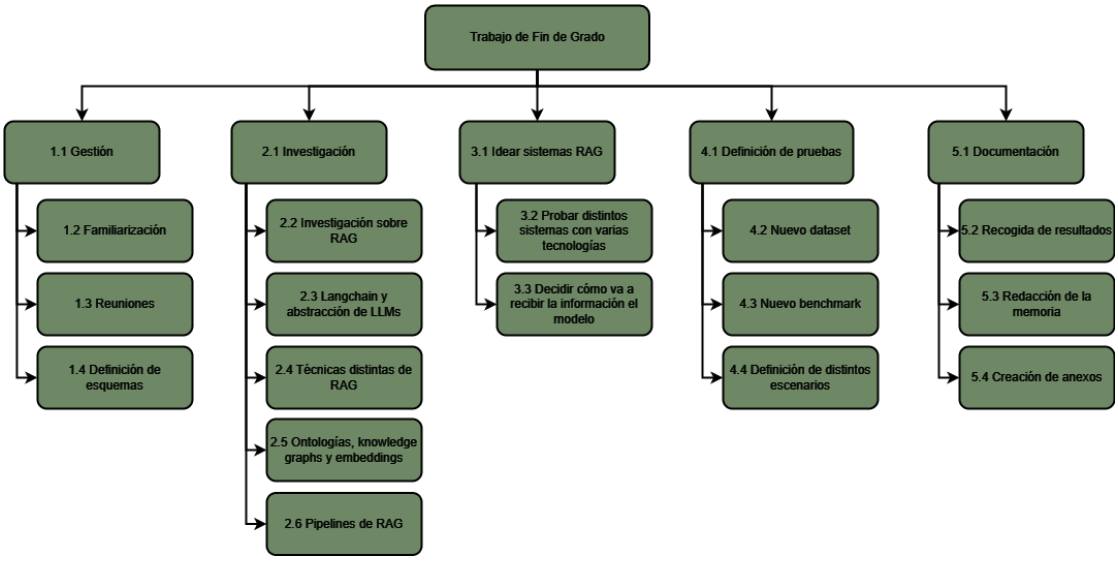

Figura 1: Esquema de división del trabajo

2.1.1. Gestión y planificación

En esta subsección se detallan las tareas llevadas a cabo con el fin de tener una buena gestión y planificación del proyecto.

Reuniones
Paquete de trabajo: Gestión
Tiempo estimado: 15 horas
Descripción: Se realizaron reuniones semanales tanto con Mikel Egaña, como con Arkaitz Carbajo, además, con Unai López se hicieron también reuniones más extensas para revisar el estado del proyecto.
Entregables: Actas de reuniones
Recursos: Microsoft Teams en las reuniones telemáticas y OneNote para la toma de notas.

Tabla 1: Reuniones

2.1.2. Investigación

Esta sección del proyecto ha sido la que más tiempo ha tomado, ya que se dedicó mucho tiempo al principio a investigar sobre el estado del arte y las tecnologías a utilizar. Además, durante la fase de desarrollo también se han realizado investigaciones para mejorar el modelo.

Investigación sobre el funcionamiento de los LLMs
Paquete de trabajo: Investigación
Tiempo estimado: 20 horas
Descripción: Se investigó sobre el funcionamiento de los LLMs, cómo se entrenan y cómo se utilizan. Además, se investigó sobre los diferentes modelos de LLMs y cuál sería el más adecuado para el proyecto.
Entregables: Notas sobre el funcionamiento de los LLMs
Recursos: OneNote para la toma de notas

Tabla 2: Investigación sobre el funcionamiento de los LLMs

Langchain y abstracción de LLMs
Paquete de trabajo: Investigación
Tiempo estimado: 20 horas
Descripción: Durante este paquete de trabajo de consultó un curso sobre RAG y Langchain, que se utilizó de base para implementar los sistemas de RAG que se evalúan en este trabajo.
Entregables: Notas sobre Langchain y abstracción de LLMs
Recursos: ChatGPT and LangChain: The Complete Developer's Masterclass [4]

Tabla 3: Langchain y abstracción de LLMs

Técnicas de RAG
Paquete de trabajo: Investigación
Tiempo estimado: 20 horas
Descripción: Se investigó sobre el estado del arte de sistemas de RAG y sobre cómo son utilizados en la actualidad. Además, se investigaron técnicas de recuperación de información con distintas bases de conocimiento.
Entregables: Notas sobre técnicas de RAG
Recursos: OneNote para la toma de notas

Tabla 4: Técnicas de RAG

Ontologías, knowledge graphs y embeddings
Paquete de trabajo: Investigación
Tiempo estimado: 30 horas
Descripción: Se investigó sobre ontologías y knowledge graphs, que se consultaron también en las reuniones con Mikel Egaña. Además, se investigaron embeddings y se hicieron pruebas para determinar qué información era útil para recuperar de una base de datos vectorial. Además, se probaron distintas bases de datos vectoriales.
Entregables: Notas sobre ontologías, knowledge graphs y embeddings. Decisión en base de datos vectorial.
Recursos: OneNote para la toma de notas. Visual Studio Code para la programación

Tabla 5: Ontologías, knowledge graphs y embeddings

Pipelines de RAG
Paquete de trabajo: Investigación
Tiempo estimado: 20 horas
Descripción: Se investigó sobre creación de pipelines con RAG, recuperando información de una base de conocimiento para máyor precisión generando JQL. Se probó a introducir en el benchmark el pipeline.
Entregables: Notas sobre RAG. Código con distintas pruebas.
Recursos: OneNote para la toma de notas. Visual Studio Code para la programación

Tabla 6: Pipelines de RAG

2.1.3. Ideas sistemas RAG

2.1.4. Definición de pruebas

Durante esta sección se definieron las pruebas que se iban a realizar para evaluar la calidad de cada una de las alternativas. Se ideó un nuevo conjunto de datos y se modificó el código del benchmark para ajustarlo al pipeline de RAG y permitir que sea replicable.

Creación de nuevo Dataset
Paquete de trabajo: Definición de pruebas
Tiempo estimado: 2 horas
Descripción: Se creó un nuevo conjunto de datos con 100 preguntas, que se utilizó para evaluar las distintas alternativas propuestas. Además, se utilizó un dataset existente en Hugging Face [5] para complementar el conjunto de datos.
Entregables: Nuevo conjunto de datos, archivo CSV.
Recursos: Visual Studio Code para la programación

Tabla 7: Creación de nuevo Dataset

Modificación del benchmark
Paquete de trabajo: Definición de pruebas
Tiempo estimado: 5 horas
Descripción: Se modificó el código del benchmark para ajustarlo al pipeline de RAG y permitir que sea replicable. Se añadieron las pruebas con el nuevo conjunto de datos y se realizaron pruebas con el nuevo conjunto de datos.
Entregables: Código modificado del benchmark
Recursos: Visual Studio Code para la programación

Tabla 8: Modificación del benchmark

subsectionPresupuesto A lo largo de esta subsección se detalla el presupuesto necesario para el estudio transcurrido en este trabajo de fin de grado. Se incluyen los costes de los recursos humanos, los costes de los recursos materiales y los costes de los recursos software.

2.1.5. Costes de software

El estudio de este trabajo se ha realizado con herramientas de software de código abierto en la medida de lo posible, si bien también se ha hecho uso de llamadas de API de OpenAI.

Como editores de código y ontologías, se han utilizado Visual Studio Code y Protégé respectivamente, ambos de código abierto y gratuitos. Para la gestión de versiones se ha utilizado Git, también de código abierto y gratuito.

2.1.6. Costes de mano de obra

Los costes de mano de obra se han calculado en base a las horas de trabajo invertidas en el proyecto y el salario medio de un ingeniero informático en España, además del coste de dos profesores adjuntos de la Universidad del País Vasco y un tutor de la empresa LKS Next-GobTech. En el caso del ingeniero informático, se asume un rol de programador junior, con un salario medio de 21000€ brutos anuales. En el caso del prfesor adjunto, según la página de la UPV/EHU, el salario medio es de 34144€ brutos anuales.

2.1.7. Costes de hardware

Para la realización de este trabajo se han utilizado tanto un portátil Lenovo Thinkpad T14, durante la estancia en LKS Next-GobTech, valorado en 900€, como un ordenador de sobremesa montado por partes, con los siguientes componentes:

- Procesador: AMD Ryzen 7 7800X3D
- Tarjeta gráfica: NVIDIA GeForce RTX 2080 Super 8GB VRAM
- Memoria RAM: 32 GB DDR5

Valorado, en el momento de compra, en 1500€.

La amortización de estos dos equipos se puede calcular en un periodo de 10 años, ya que ambos equipos tienen una vida útil de al menos 10 años a día de hoy.

3. Tecnologías

A continuación se detallarán las distintas tecnologías que serán estudiadas durante este TFG. Cabe recalcar que varias de estas distintas tecnologías propuestas, como los grafos de conocimiento o las ontologías, han requerido de un estudio previo para poder ser implementadas en el proyecto.

Independientemente de los resultados que se obtengan con cada una de ellas, es necesario tener en cuenta el proceso de familiarización con las mismas, así como el tiempo invertido en su estudio y posterior implementación para un desempeño óptimo.

3.1. Modelos grandes de lenguaje

Dentro del campo de la inteligencia artificial y el Procesamiento del Lenguaje Natural (PLN), los modelos grandes de lenguaje, conocidos en inglés como Large Language Model (LLM) han sido una de las tecnologías más revolucionarias de los últimos años. Estos modelos son capaces de aprender de grandes cantidades de texto y generar texto de manera coherente y con sentido, pudiendo así responder a preguntas basándose en el contexto proporcionado.

Los LLMs se basan en arquitecturas de redes neuronales profundas, como los *transformers* [6], que permiten procesar secuencias de texto de manera más eficiente. Gracias a su mecanismo de atención, el cual permite al modelo enfocarse en las partes más relevantes de la secuencia de texto, los transformers han sido la base de muchos de los modelos de lenguaje más grandes y potentes de la actualidad.

A diferencia de modelos lingüísticos anteriores, los LLMs son capaces de aprender de manera no supervisada, lo que les permite obtener información de grandes cantidades de texto sin necesidad de etiquetas. Esto ha permitido el desarrollo de modelos masivos, como GPT-3 [7], que han demostrado ser capaces de realizar tareas de generación de texto, traducción, resumen, entre otras, con resultados sorprendentes.

3.2. Retrieval Augmented Generation

Se conoce como Retrieval Augmented Generation (RAG) a la arquitectura que combina la recuperación de información con la generación de texto. Esta arquitectura se compone de dos partes principales: un modelo de recuperación y un modelo de generación. El modelo de recuperación se encarga de recuperar información relevante de una base de conocimiento, mientras que el modelo de generación se encarga de generar texto basado en la información recuperada.

Esta arquitectura es especialmente útil cuando se trabaja con modelos de lenguaje grandes, ya que mejora el problema de las alucinaciones. En lugar de generar respuestas en base al conocimiento del que disponen durante el entrenamiento, que puede dar resultados erróneos, el modelo puede acceder a bases de conocimiento factual con las que puede generar respuestas más precisas y acordes al contexto.

3.2.1. Funcionamiento

El funcionamiento típico de esta arquitectura consta de un flujo dividido en dos partes principales, la recuperación de contexto y la generación de respuestas, que se explicarán brevemente a continuación:

Durante la recuperación de contexto se consulta en una base de conocimiento, que podría ser una base de datos vectorial, un grafo de conocimiento o una ontología, entre otros. Para hacer una consulta, se ha de contrastar la pregunta que un usuario haga con la información contenida, para obtener la información más relevante posible. Esta tarea requiere gran atención ya que es crucial de cara al desempeño que vaya a lograr el sistema.

Una vez se ha recuperado la información, se pasa a la generación de respuestas. En esta etapa, se utiliza la información recuperada junto con la pregunta inicial para guiar al modelo de lenguaje en la generación de respuestas. De esta manera, el modelo puede generar respuestas más precisas y acordes al contexto proporcionado.

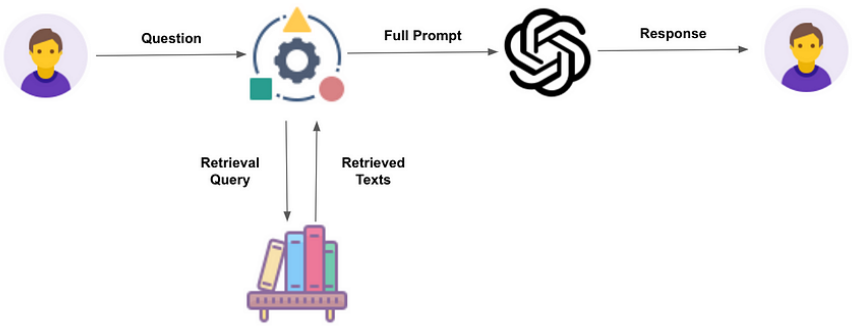

Figura 2: Esquema de funcionamiento de una arquitectura RAG.

3.3. Word embeddings

Los word embeddings, o simplemente embeddings, son una representación de palabras en un espacio multidimensional y son un técnica fundamental dentro del campo del PLN. Consiste en la transformación de palabras en vectores de nú-

meros reales dentro del espacio, de manera que se puedan realizar operaciones matemáticas con ellas.

Para transformar las palabras en vectores se busca capturar la semántica de las palabras, de manera que palabras similares tengan vectores cercanos. Los métodos tradicionales de embeddings, como *word2vec* [8] o *GloVe* [9], utilizan modelos de aprendizaje no supervisado para aprender la representación de las palabras a partir de grandes cantidades de texto.

Los embeddings tienen muchas aplicaciones, como la clasificación de texto, la traducción automática o la recuperación de información, además, son una técnica que a día de hoy se sigue estudiando y mejorando.

3.4. Ontologías

Una ontología en el campo de la informática es una especificación formal y explícita de una conceptualización. En otros términos, una ontología es una representación de un conjunto de conceptos dentro de un dominio y las relaciones entre esos conceptos.

La utilidad de las ontologías radica en que permiten a los sistemas de información compartir conocimiento de manera más eficiente, ya que proporcionan una estructura común y unificada para la representación de los datos. Además, las ontologías permiten la interoperabilidad entre sistemas, ya que facilitan la comunicación entre ellos al tener una representación común del conocimiento. A día de hoy, son objeto de estudio en combinación con los modelos de lenguaje, explorando la posibilidad de mejorar la interpretación de los datos y la generación de consultas.

El lenguaje usado para definir ontologías es el *Web Ontology Language* (OWL), que es un lenguaje de marcado semántico para publicar y compartir ontologías en la web. OWL es desarrollado por el *World Wide Web Consortium* (W3C) y es una extensión de *Resource Description Framework* (RDF), que es un modelo de datos basado en grafos para describir recursos web.

3.5. Grafos de conocimiento

Los grafos de conocimiento, conocidos más comúnmente como *knowledge graphs* en inglés, son estructuras de datos que representan información de manera semántica. Estos grafos se componen de nodos y aristas, donde los nodos representan entidades y las aristas representan relaciones entre estas entidades. Los grafos de conocimiento son especialmente útiles para representar información estructurada y para realizar consultas complejas sobre esta información.

Los grafos de conocimiento se popularizaron con la creación de *Google Knowledge Graph* en 2012, un servicio de Google que proporciona información adicional en los resultados de búsqueda. Este servicio se basa en un grafo de conocimiento que contiene información sobre una gran cantidad de entidades y relaciones entre estas entidades. Desde entonces, los grafos de conocimiento han sido utilizados en una gran variedad de aplicaciones, como la búsqueda semántica, las redes sociales y la recomendación de contenido.

A día de hoy, existen dos modelos de grafo de conocimiento más populares: *Resource Description Framework* (RDF) y *Property Graph Model*. El modelo RDF es un modelo de grafo de conocimiento basado en la web que se utiliza para representar información semántica en la web. En este tipo de grafo, los nodos representan recursos atómicos y las aristas representan relaciones entre estos, además, estos grafos son dirigidos, en este tipo de grafos la unidad de discurso se considera la unión de dos nodos por una relación. Por otro lado, el modelo de *Property Graph* es un modelo de grafo de conocimiento que tiende a usar nodos como objetos de un dominio, como tipos de datos ricos, donde estos mismos pueden considerarse la unidad de discurso, como se explica en esta cuestión de Stack Overflow [10].

En el ámbito de la inteligencia artificial, los grafos de conocimiento son especialmente útiles para mejorar el rendimiento de los modelos de lenguaje. Al combinar la información de un grafo de conocimiento con la información de un modelo de lenguaje, se pueden generar respuestas más precisas y acordes al contexto. Esto es una ventaja sobre los *Embeddings* ya que proporcionan un mayor contexto y una mayor precisión en las respuestas generadas. Son especialmente útiles en contextos donde es relevante mucha información interrelacionada, como en ámbitos de salud o medicina, donde es realmente útil tener en cuenta toda la información disponible para dar una respuesta precisa.

4. Diseño

Durante esta sección se hablará del punto de partida del asistente JiraGPT Next y del diseño que se ha propuesto como mejora para la precisión de las consultas generadas. Se presentarán 3 alternativas utilizando arquitecturas RAG, que buscan dotar al modelo de información sobre la generación de consultas JQL.

4.1. Estado inicial

Para cualificar el estado inicial del modelo se ha de poner en contexto la técnica utilizada para evaluar la precisión: un *benchmark* de 70 preguntas en el que se relaciona cada una con las incidencias que deberían ser recuperadas por el modelo.

La manera en la que se evalúa es ejecutando el conjunto entero de preguntas y evaluando si el asistente ha recuperado exactamente las incidencias contenidas en el conjunto de datos. Esto se decidió de esta manera ya que puede darse el caso en el que diferentes consultas devuelvan las mismas incidencias, lo que se consideraría correcto, con tal de que esas incidencias respondan a la pregunta del usuario.

En el momento de inicio de este trabajo, el asistente JiraGPT Next oscilaba entre un 45 y un 50 % de precisión en la recuperación de incidencias. Este resultado es fruto de una investigación sobre *prompt engineering* realizada previamente por Joel García [1]. El objetivo entonces, es buscar nuevas maneras de mejorar la precisión ofrecida por el modelo.

4.2. Propuestas

Se propone utilizar arquitecturas RAG para mejorar la precisión, ofreciendo al modelo información sobre la generación de consultas Jira Query Language (JQL). La idea detrás de esto es que, al tener un modelo de recuperación que pueda acceder a una base de conocimiento, el modelo de generación pueda generar respuestas más precisas y acordes al contexto proporcionado. Además, se propone un nuevo conjunto de datos:

Si bien el conjunto de datos inicial era robusto, se ha propuesto un nuevo conjunto, de 100 preguntas, que busca, no solo tener más datos, sino hacerlos más diversos y cambiar en cierto modo las preguntas para cubrir el máximo número de casos posible. Este conjunto de datos se ha pensado durante el desarrollo y las diferentes pruebas lanzadas y también se ha usado como apoyo un dataset existente en *Hugging Face* [5].

A continuación, se describirán las distintas alternativas propuestas para mejorar la precisión del modelo JiraGPT Next.

4.2.1. Ontología

Durante el inicio de este trabajo se consultaron artículos como *Sequeda et al.* [11], que exploraban la posibilidad de utilizar ontologías en el prompt para mejorar la interpretación de los datos y la generación de consultas SQL, logrando resultados prometedores. Partiendo de esta idea, se propone entonces crear una ontología que represente las reglas que existen en las consultas JQL. La información que se pretende representar en la ontología se ha extraído directamente de la documentación oficial de JIRA, brindada por Atlassian, donde se detallan las reglas que se deben seguir para la creación de consultas JQL [12]. Esta ontología serviría para interpretar las reglas que hay que seguir al generar consultas JQL, además, consta de ejemplos en cada una de las clases definidas, que ayuda a comprender mejor el funcionamiento de las reglas.

La ontología se ha desarrollado siguiendo el estándar *Web Ontology Language* (OWL), que es un lenguaje de marcado semántico para publicar y compartir ontologías en la web. OWL es desarrollado por el *World Wide Web Consortium* (W3C) y es una extensión de *Resource Description Framework* (RDF).

Durante esta parte, se considera una ejecución del benchmark como *baseline* para comparar los resultados obtenidos con la ontología y sin ella. Esta primera ejecución se realizaría inyectando el archivo entero de la ontología en el prompt, de manera que el modelo pueda acceder a la información de la ontología y utilizarla para generar consultas más precisas.

Esta aproximación podría presentar resultados que, a priori, parecieran prometedores. Sin embargo, de cara al coste de inyectar un prompt tan grande, no es viable en un entorno de producción. Por ello, se propone una recuperación de información de la ontología dado un campo relevante para la pregunta del usuario. El diagrama en la siguiente figura muestra cómo será la interacción entre el modelo y la ontología.

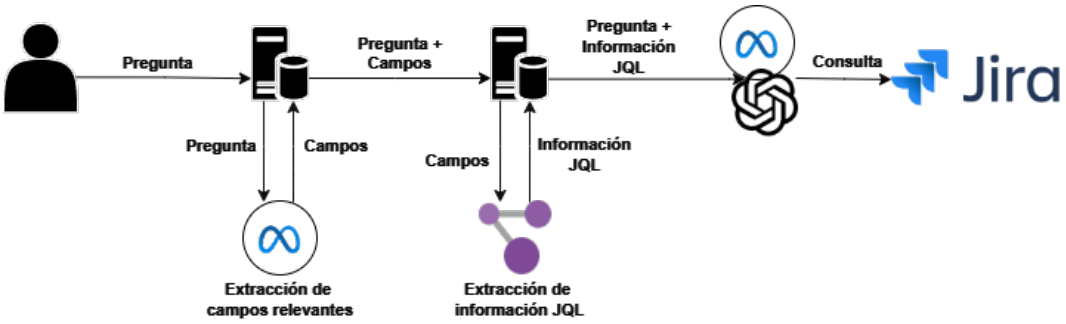

Figura 3: Diagrama de RAG con ontología

Si bien una ontología es una manera de representar un espacio de conocimiento, la documentación de JIRA no presenta una estructura demasiado compleja que representar con la semántica de una ontología. Podría considerarse que, por ende, no es realmente necesario el uso de una ontología para representar las reglas de JQL. Sin embargo, tras lo realizado en este trabajo, no sería complicado utilizar otra ontología que reuniese información más compleja y permitiese explotar la semántica de la ontología para mejorar la generación de consultas JQL.

4.2.2. Embeddings

De igual manera que en el diseño de la ontología, se propone capturar en esta base de datos vectorial la documentación oficial de JIRA. Esta base de conocimiento prueba a guardar los embeddings de los diferentes descripciones dentro de la documentación. La información que se ha considerado relevante ha sido cada campo/operador/función y sus respectivos ejemplos. De esta manera, dada una pregunta se podrían obtener los embeddings de las palabras y compararlas con los embeddings de la base de datos, de manera que el modelo obtenga ejemplos de campos relevantes para la pregunta del usuario.

Para capturar toda la información de la documentación se ha diseñado un programa en Python que realiza técnicas de *web scraping* y recorre las entradas de la documentación extrayendo de las tablas los ejemplos. Una vez hecho esto, se guarda en un archivo de texto que será dividido en diferentes partes para ser procesado por el modelo de embeddings.

Teniendo la pregunta del usuario, se ha de traducir al inglés, ya que la documentación oficial está en inglés y dada una pregunta en castellano la recuperación de información de los embeddings no sería efectiva. Para traducir la pregunta, se utiliza LLama3 y, dada la pregunta traducida, se hace una búsqueda en la base de datos vectorial, obteniendo los ejemplos más relevantes para la pregunta dada. Esta información se inyecta en el prompt para que el modelo pueda generar una respuesta más precisa. El diagrama en la siguiente figura muestra cómo será la interacción entre el modelo y la base de datos vectorial.

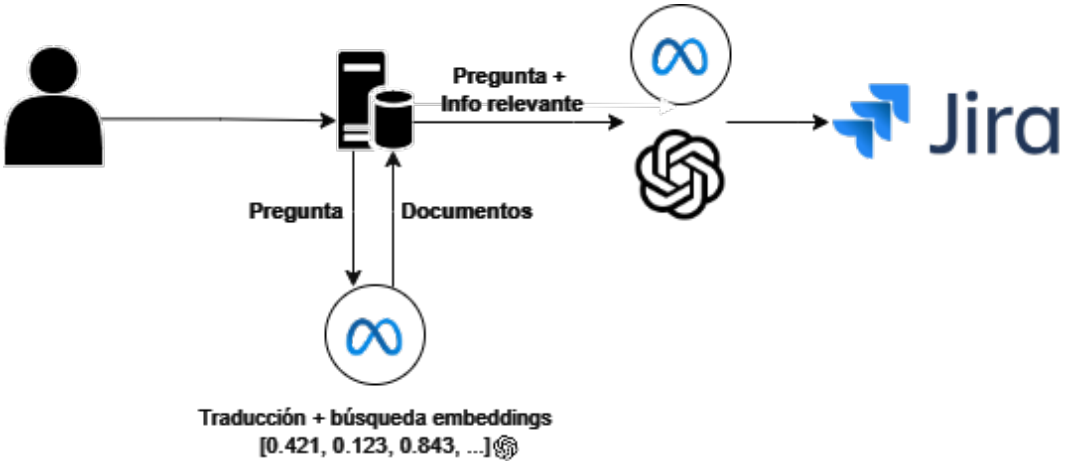

Figura 4: Diagrama de RAG con embeddings

4.2.3. Grafos de conocimiento

Como última propuesta para el trabajo, se ha planteado el uso de grafos de conocimiento para tratar de guiar al modelo en la estructura de la información contenida en la plataforma JIRA de LKS Next-GobTech. La idea detrás de esta propuesta es que, al tener información proveniente de un grafo de conocimiento, el modelo sea capaz de generar respuestas más robustas y acordes al contexto. Para ello, se ha diseñado un grafo de conocimiento que representa un proyecto de JIRA.

El flujo sería el representado en la figura siguiente, donde se muestra cómo, dada una pregunta de un usuario se extrae información del grafo de conocimiento que se inyectaría en el prompt para que el modelo pueda generar una respuesta con más contexto.

5. Implementación

A lo largo de esta sección se va a detallar la implementación de las distintas propuestas tecnológicas que se han descrito en el apartado anterior.

Para la implementación y el lanzamiento de muchas pruebas, se ha decidido ejecutar un LLM de manera local, ya que, de esta, manera, reducimos el coste de las llamadas a OpenAI a tan solo aquel que acarree el cálculo de los embeddings.

El modelo a utilizar será Llama3:8b, ejecutado en local mediante Ollama [1] y al cual se harán llamadas a traves de programas Python, utilizando la librería Langchain [2].

5.1. Implementación de RAG: Embeddings

Esta propuesta consta de una base de datos vectorial que contiene los embeddings de la documentación oficial de JIRA. Como software se ha optado por ChromaDB [3], una base de datos vectorial de código abierto que permite realizar búsquedas por similitud de vectores que, además, tiene una sencilla implementación con Langchain.

5.1.1. Extracción de datos

Para extraer los datos pertinentes de la documentación oficial de JIRA, se ha construido un *web scraper* que recorre la documentación oficial de JIRA en el lenguaje Python que extrae de las tablas para cada entrada título, descripción y ejemplos. Estos datos se almacenan en un archivo de texto que posteriormente será dividido en diferentes partes para ser procesado por el modelo de embeddings.

Como modelo de embeddings se han utilizado los embeddings de OpenAI, que mediante una llamada con texto devuelven el vector a almacenar en la base de datos. Todo esto con un programa en Python que utiliza Langchain para abstraer el uso de ChromaDB y de la API de OpenAI.

5.1.2. Interacción con el modelo

Cuando una pregunta es recibida en el sistema, esta será procesada por los embeddings de OpenAI, lo que genera un vector que se compara con los vectores almacenados en la base de datos vectorial. Los vectores más cercanos a la pregunta serán devueltos, conteniendo la información de Jira pertinente. Esta información

[1] https://ollama.com
[2] https://www.langchain.com/
[3] https://www.trychroma.com/

se inyecta en el prompt para que el modelo pueda generar una respuesta más precisa.

5.2. Implementación de RAG: Ontología

Como se ha mencionado en el apartado de diseño, se ha construido una ontología desde 0 utilizando la documentación oficial de JIRA para crear consultas JQL. Esta ontología describe los distintos tipos de operadores, campos y funciones que existen en este lenguaje de consulta. Además, se han añadido las relaciones entre estos elementos para que el modelo pueda inferir la información necesaria para responder a las preguntas. La idea detrás de esta ontología es modelar cómo funciona Jira Query Language, para que el modelo cometa menos errores de interpretación y pueda responder de manera más precisa. Se ha desarrollado tras el análisis de la documentación y la realización de consultas con el tutor para describir de manera correcta una ontología, siguiendo la semántica necesaria.

Esta ontología se ha desarrollado utilizando el software Protégé [13], gratuito y de código abierto, desarrollado por la universidad de Stanford. Una vez creada, se puede exportar como archivo RDF, para interactuar con ella desde fuera de Protégé.

5.2.1. Interacción con el modelo

La interacción posible de la ontología con el modelo parte desde la inyección del archivo RDF entero, como describe Sequeda et al. [11]. Sin embargo, esta aproximación no es viable en un entorno de producción, ya que inyectar un archivo tan grande en el prompt no es eficiente, pero se conserva de baseline.

Para obtener información relevante de la ontología se ha utilizado un LLM que extrae los campos relevantes dada una pregunta, que son expuestos anteriormente en el prompt, ya que son campos descritos en la ontología que no conoce el modelo. La respuesta del modelo serán los 3 campos más relevantes de entre todos los descritos en el prompt y desde los que se realizará una consulta a la ontología para obtener la información relevante de la misma.

Esta información obtenida consistirá en los campos, operadores, ejemplos, descripción y subclases (de existir) que serán inyectados en el prompt para que el modelo pueda generar una respuesta partiendo de un mayor contexto.

Referencias

[1] Joel García Escribano. «JiraGPT Next». En: (2023).

[2] Frederick P. Brooks. *The mythical man-month – Essays on Software-Engineering*. Addison-Wesley, 1975.

[3] Scott Chacon y Ben Straub. *Pro git*. Apress, 2014.

[4] Stephen Grider. *ChatGPT and LangChain: The Complete Developer's Masterclass*. Accessed: During February 2024. URL: https://www.udemy.com/course/chatgpt-and-langchain-the-complete-developers-masterclass/.

[5] @ManthanKulakarni. *Text2JQL*. Accessed: during March 2024. URL: https://huggingface.co/datasets/ManthanKulakarni/Text2JQL_v2.

[6] Ashish Vaswani et al. «Attention is All you Need». En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[7] Tom Brown et al. «Language Models are Few-Shot Learners». En: *Advances in Neural Information Processing Systems*. Ed. por H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, págs. 1877-1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac14
Paper.pdf.

[8] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[9] Jeffrey Pennington, Richard Socher y Christopher Manning. «GloVe: Global Vectors for Word Representation». En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. por Alessandro Moschitti, Bo Pang y Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, oct. de 2014, págs. 1532-1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[10] @FrobberOfBits. *Graph databases vs triple stores*. Accessed: 2023-05-22. URL: https://stackoverflow.com/questions/30166006/graph-databases-vs-triple-stores-when-to-use-which.

[11] Juan Sequeda, Dean Allemang y Bryon Jacob. *A Benchmark to Understand the Role of Knowledge Graphs on Large Language Model's Accuracy for Question Answering on Enterprise SQL Databases*. 2023. arXiv: 2311.07509 [cs.AI].

[12] Atlassian. *Use advanced search with Jira Query Language (JQL)*. Accessed: 2024-03-30. URL: https://support.atlassian.com/jira-service-management-cloud/docs/use-advanced-search-with-jira-query-language-jql/.

[13] Stanford University. *Protégé*. Accessed: 2024-03-20. URL: https://protege.stanford.edu/.