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

ChatGPT-RAG para una mejor interacción con JIRA

Estudiante: Aróstegui García, Alberto

Directores: Egaña Aranguren, Mikel; López Novoa, Unai

Curso: 2023-2024

Fecha: 15 de julio de 2024

Resumen:
Este trabajo presenta un estudio de técnicas para mejorar la experiencia de usuario en la gestión de proyectos. Se utiliza como base Jira, una de las herramientas más usadas con este fin y que, aunque presenta una interfaz de usuario intuitiva, tiene limitaciones para obtener datos complejos. Para salvar esta limitación, Jira ofrece Jira Query Language (JQL), un lenguaje de consultas que permite obtener datos con mucha mayor precisión, pero que presenta complejidades en su uso.

Para salvar esta limitación, este trabajo plantea aplicar técnicas de Generación Aumentada por Recuperación (GAR) para crear un interfaz que esconda las complejidades de JQL al gestor de proyectos. Se plantea un estudio 3 técnicas de GAR (Ontologías, Embeddings y Grafos de Conocimiento) combinadas con Modelos Grandes de Lenguaje. Se presenta una evaluación realizada con 4 proyectos Jira proporcionados por la empresa LKS Next-GobTech y una batería de 100 preguntas.

Palabras Clave: *RAG, LLM, Jira, JQL, Ontología, Knowledge Graph, Embeddings*

Laburpena:
Lan honek proiektuen kudeaketa esparruan erabiltzaileen esperientzia hobetzeko tekniken azterketa bat aurkezten du. Oinarri bezala Jira erabiltzen da, helburu honetarako gehien erabiltzen den tresnetako bat. Honek, erabiltzaile-interfaze intuitibo bat izan arren, datu konplexuak lortzeko mugak ditu. Muga hauek saihesteko, Jirak Jira Query Language (JQL) eskaintzen du: datuak zehaztasun handiagoarekin lortzeko aukera ematen duen kontsulta-lengoaia, baina erabiltzeko konplexutasunak dituena.

Egoera hau hobetu nahian, lan honek Berreskurapenaren bidez Handitutako Sorkuntza (BHS) teknikak aplikatzea proposatzen du. Hauek JQL-ren konplexutasunak ezkutatzen dituen interfaze bat sortzeko oinarri izan daitezke. Lan honetan 3 BHS teknika aztertzen dira (Ontologiak, Embedding-ak eta Ezagutza Grafoak), Hizkuntza Eredu Handiekin batera erabiltzen direnak. Tekniken ebaluazioa LKS Next-GobTech enpresak emandako 4 Jira proiektuekin eta 100 galderako bateria batekin egiten da.

Gako-hitzak: *BHS, HEH, Jira, JQL, Ontologiak, Ezagutza Grafoak, Embeddingak*

Abstract:

This work presents a study of techniques to improve the user experience in project management. Jira, one of the most widely used tools for this purpose, is used as a base. Although Jira features an intuitive user interface, it has limitations

in obtaining complex data. To overcome this limitation, Jira offers Jira Query Language (JQL), a query language that allows for much more precise data retrieval, but it presents complexities in its use.

To address this limitation, this work proposes applying Retrieval-Augmented Generation (RAG) techniques to create an interface that hides the complexities of JQL from the project manager. A study of three RAG techniques (Ontologies, Embeddings, and Knowledge Graphs) combined with Large Language Models is proposed. An evaluation is presented, conducted with 4 Jira projects provided by the company LKS Next-GobTech and a total of 100 questions.

Key Words: *RAG, LLM, Jira, JQL, Ontology, Knowledge Graph, Embeddings*

Índice

Índice de figuras

Índice de tablas

Abreviaturas

API Application Programming Interface

JQL Jira Query Language

LLM Large Language Model

RAG Retrieval Augmented Generation

PLN Procesamiento del Lenguaje Natural

KG Knowledge Graph

RDF Resource Description Framework

RDFS Resource Description Framework Schema

OWL Web Ontology Language

SPARQL SPARQL Protocol and RDF Query Language

1. Introducción

En la actualidad, la gestión eficiente de proyectos es esencial para las empresas de desarrollo de software que buscan mantenerse competitivas y ofrecer productos de alta calidad. En este contexto, las herramientas que facilitan la organización, planificación y control de recursos se convierten en elementos clave. Este trabajo, realizado en colaboración con la empresa LKS Next-GobTech, que se dedica al desarrollo de software, con gran enfoque en la innovación, explora estas tecnologías.

El proyecto se basa en mejorar las capacidades de JIRA, una herramienta ampliamente utilizada para la gestión de proyectos, mediante la implementación de tecnologías avanzadas de procesamiento del lenguaje natural (PLN). En el ámbito de la gestión de proyectos, la persona encargada de la supervisión y control de las tareas necesita acceder a información específica sobre el estado de los proyectos, como el número de incidencias abiertas, el tiempo estimado para completar una tarea o el progreso de un proyecto en particular. Para obtener esta información, JIRA ofrece un lenguaje de consulta llamado JIRA Query Language (JQL), que permite realizar consultas avanzadas sobre los datos almacenados en la plataforma. Sin embargo, un gestor de proyectos no siempre está familiarizado con la sintaxis y estructura de JQL, lo que puede dificultar la obtención de la información deseada. Es por esto que, partiendo del trabajo previo de un compañero, que desarrolló un asistente conversacional capaz de generar consultas JQL a partir de preguntas en lenguaje natural, este proyecto explora la integración de una arquitectura de Generación Aumentada por Recuperación (RAG, por sus siglas en inglés) para aumentar la precisión y utilidad de las respuestas ofrecidas por el sistema, buscando mejorar la interacción y eficiencia en el uso de JIRA.

La estructura del trabajo abarca desde la contextualización de las herramientas y metodologías utilizadas en la gestión de proyectos hasta la descripción detallada del desarrollo e implementación de la solución propuesta. Se analiza cómo las tecnologías de modelos de lenguaje de gran escala (LLM) y la generación aumentada por recuperación pueden integrarse en el flujo de trabajo existente para mejorar la interacción y eficiencia en el uso de JIRA.

1.1. Objetivos del trabajo

Para la realización de este trabajo, se han establecido unos objetivos junto con LKS Next-GobTech.

El objetivo principal de este trabajo es realizar un estudio sobre la potencial mejora de precisión en la generación de consultas JQL que ofrecen los modelos de lenguaje natural de hoy en día. Para ello, se propone la integración de una ar-

quitectura de Generación Aumentada por Recuperación (RAG) en un asistente conversacional previamente desarrollado, que permita mejorar la interacción con JIRA y la eficiencia en la obtención de información relevante.

De cara a la realización de las pruebas y la evaluación de resultados obtenidos se ha desarrollado en Python un entorno de pruebas que permita evaluar la eficacia de la solución propuesta. Este entorno de pruebas se ha diseñado para realizar pruebas de rendimiento y comparar los resultados obtenidos con y sin la integración de RAG. Además, permite de manera sencilla la evaluación de más modelos de lenguaje y diferentes técnicas de recuperación de información.

2. Contexto

Este trabajo de fin de grado responde a las necesidades de la empresa LKS Next-GobTech[1], una empresa de desarrollo de software con enfoque en la innovación. LKS Next-GobTech es una empresa que desarrolla software a medida para sus clientes, principalmente la adminstración pública del País Vasco y, en menor medida, empresas privadas. Utilizan técnicas de desarrollo ágil y metodologías de gestión de proyectos para llevar a cabo sus proyectos.

Para comprender las necesidades de la empresa que este trabajo pretende solucionar, primero se han de poner en contexto las herramientas y metodologías que utilizan. 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 y cuyo objetivo era dar una visión de las posibilidades de los LLMs en el desarrollo de software y proporcionar a LKS una herramienta para el futuro desarrollo de sus proyectos, 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.

2.1. 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 controvertido 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 vista de lo expuesto, resulta 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] https://www.lksnext.com/es/servicios/tecnologia/gobtech/outsourcing-it-estrategico-con-lks-next/

2.2. Herramientas

2.2.1. 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 el resultado de la división del proyecto en 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.

2.2.2. 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.

2.3. 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. JQL es un lenguaje de consulta que permite recuperar incidencias de Jira de manera avanzada y precisa, pero que requiere conocimientos técnicos para su uso.

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.

Se puede apreciar una captura de la aplicación en la figura 1, que consiste en una ventana de chat en la que se puede interactuar con el asistente con varias opciones disponibles. La persona que lo utilizase, presumiblemente un gestor de proyectos que no conoce JQL y necesita información sobre el proyecto en Jira podría lanzar sus preguntas y obtener respuestas con un resumen y los campos que ha solicitado. Tendría varias opciones, como la temperatura, que representa lo 'creativo' que puede ser el modelo (más estocástico o determinista), la plantilla o el modelo a utilizar y un checkbox para activar la pregunta compleja, que seleccionaría los campos relevantes de la respuesta a la consulta JQL.

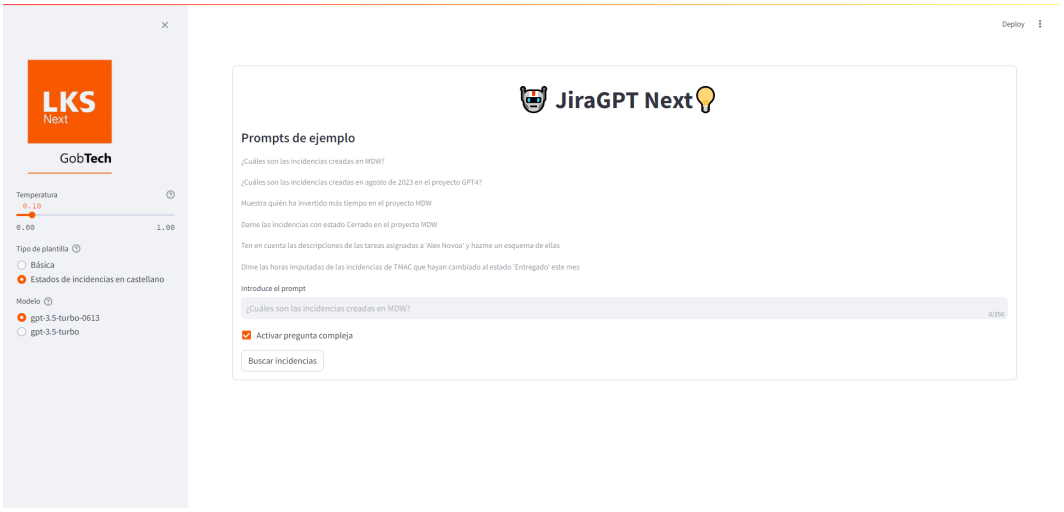

Figura 1: Interfaz de la aplicación

La idea de este nuevo trabajo es realizar un estudio del potencial cambio en precisión que se puede lograr utilizando técnicas de Retrieval Augmented Generation (RAG), entendiendo precisión como el número de preguntas que el sistema es capaz

de responder correctamente. Para ello, se van a proponer tres implementaciones diferentes de una base de conocimiento de la que el modelo pueda aprovechar la información contenida para generar respuestas partiendo de un mayor contexto.

3. 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.

3.1. Tareas

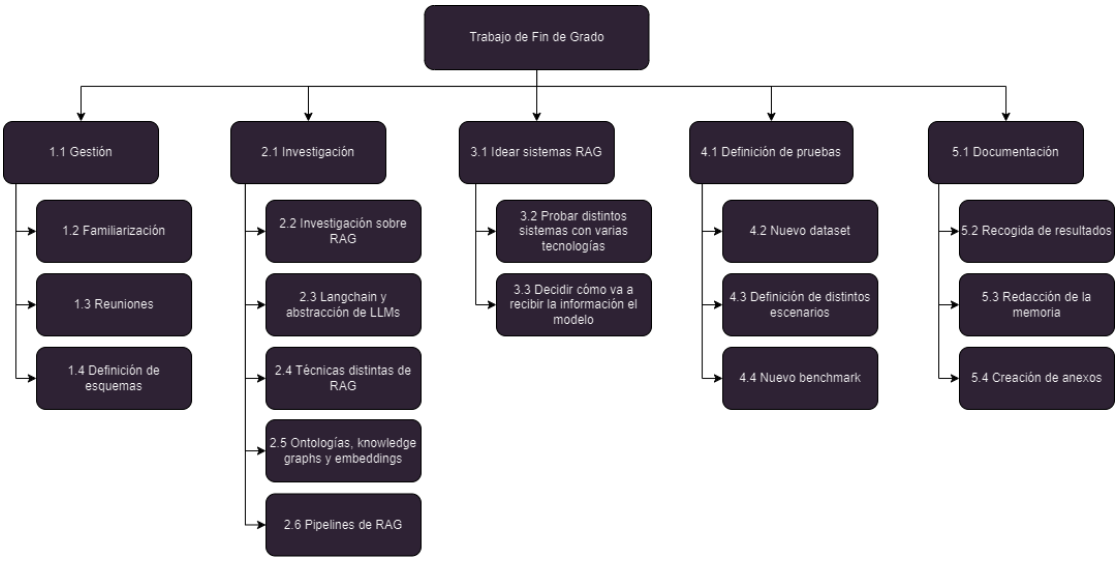

Figura 2: Esquema de división del trabajo.

3.1.1. 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.

3.1.2. 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.

3.1.3. Idear sistemas RAG

Probar distintos sistemas con varias tecnologías.
Paquete de trabajo: Idear sistemas RAG
Tiempo estimado: 50 horas
Descripción: Se idearon distintos sistemas de RAG con distintas tecnologías, como ontologías, knowledge graphs y embeddings. Se probaron distintas implementaciones de estas tecnologías para ver cuál era la más adecuada para el proyecto.
Entregables: Código con distintas pruebas
Recursos: Visual Studio Code para la programación

Tabla 7: Probar distintos sistemas con varias tecnologías.

Decidir cómo va a recibir la información el modelo.
Paquete de trabajo: Idear sistemas RAG
Tiempo estimado: 20 horas
Descripción: Decidir, para cada implementación, cómo podría recibir la información el modelo, cómo extraerla de cada base de conocimiento y cómo presentarla.
Entregables: Código con distintas pruebas
Recursos: Visual Studio Code para la programación

Tabla 8: Decidir cómo va a recibir la información el modelo.

3.1.4. 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 9: Creación de nuevo Dataset.

Definición de distintos escenarios.
Paquete de trabajo: Definición de pruebas
Tiempo estimado: 5 horas
Descripción: Se definieron distintos escenarios para evaluar las distintas alternativas propuestas. Se evaluó cómo debería ser el entorno de pruebas de Jira sobre el que se debería trabajar, con intención de aproximar el estudio a la realdiad en una empresa. Se definieron los criterios de evaluación y se crearon las pruebas para cada uno de los escenarios.
Entregables: Ninguno
Recursos: Visual Studio Code para la programación

Tabla 10: Definición de distintos escenarios.

Nuevo 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 11: Nuevo benchmark.

3.1.5. Documentación

Esta sección describe cómo se realizó el proceso de documentación, tras la revisión de notas tomadas a lo largo del proyecto y la consulta con los tutores.

Recogida de resultados.
Paquete de trabajo: Documentación
Tiempo estimado: 10 horas
Descripción: Se lanzan los distintos escenarios definidos y se recogen los resultados de las pruebas realizadas.
Entregables: Resultados de las pruebas
Recursos: Visual Studio Code para la programación

Tabla 12: Recogida de resultados.

Redacción de la memoria.
Paquete de trabajo: Documentación
Tiempo estimado: 50 horas
Descripción: Teniendo en cuenta la toma de notas a lo largo de todo el proyecto, la valoración de distintas aproximaciones y la consulta con tutores, se redactó la memoria acorde a lo que se había planeado al principio del proyecto
Entregables: Memoria del trabajo de fin de grado.
Recursos: Visual Studio Code para la redacción de la memoria. Compilador LaTeX.

Tabla 13: Redacción de la memoria.

Creación de anexos.
Paquete de trabajo: Documentación
Tiempo estimado: 5 horas
Descripción: Se crearon los anexos de la memoria, que incluyen el código del benchmark y los resultados de las pruebas realizadas.
Entregables: Anexos de la memoria.
Recursos: Visual Studio Code para la redacción de la memoria. Compilador LaTeX.

Tabla 14: Creación de anexos.

3.2. Presupuesto

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. El proyecto ha tenido una duración de 4 meses, por lo que se ha calculado el coste total en base a este periodo de tiempo.

3.2.1. 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.

Durante las 2 primeras semanas del proyecto se realizó un curso de Udemy sobre ChatGPT and LangChain, que tuvo un coste de 14,99€ [2].

El uso de la API de OpenAI conlleva un coste según el uso de la misma, para cada modelo existe un coste detallado en la página de OpenAI [3]. El coste total para este proyecto ha sido de 25€.

3.2.2. 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 [4].

Descontando impuestos y retenciones de los sueldos brutos para el programador junior, 12,20 % de IRPF: $21000 \times \frac{12{,}20}{100} \approx 2562$€ y la aportación de 1333€ a la

[2] https://www.udemy.com/course/chatgpt-and-langchain-the-complete-developers-masterclass/

[3] https://platform.openai.com/pricing

[4] https://www.ehu.eus/es/web/gardentasun-ataria/ordainketak

Seguridad Social, resultaría en un salario neto de 17105€ netos al año.

$$\text{Coste total programador junior: } \frac{17105}{12\ meses} \times 4 \approx 5701€$$

Para el tutor de empresa se ha estimado el salario para un ingeniero informático manager de proyectos, con un salario medio de 42000€ brutos anuales [6]. Descontando una aportación de IRPF de 20,14 %: $42000 \times \frac{20{,}14}{100} \approx 8463€$ y una aportación de 2667€ a la Seguridad Social, resultaría en un salario neto de 30872€ netos al año. Con el tutor de empresa se ha estado 3 meses de los 4 de duración.

$$\text{Coste total tutor de empresa: } \frac{30872}{12\ meses} \times 3\ meses \approx 7718€$$

En cuanto a los tutores de la universidad, se ha asumido un salario de 34144€ brutos anuales, descontando impuestos y retenciones, 12,20 % de IRPF: $34144 \times \frac{17{,}88}{100} \approx 6103€$ y la aportación de 2168€ a la Seguridad Social, resultaría en un salario neto de 25872€ netos al año. Los tutores han estado disponibles un total de 15 horas al mes para el proyecto.

$$\text{Coste total tutores de la universidad:}$$
$$\frac{25872}{12\ meses} \times \frac{15\ horas}{176\ horas/mes} \times 4\ meses \times 2\ tutores \approx 1470€.$$

El coste total de la mano de obra sería la suma de los costes de los tres roles, resultando en 14889€.

3.2.3. 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 1000€, 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 al ser equipos de alta gama.

La suma de ambos equipos es de 2500€. Por lo tanto, el coste para un proyecto de 4 meses es el siguiente:

$$\text{Coste total hardware: } \frac{2500}{10\ años} \times \frac{4\ meses}{12\ meses} \approx 83,33€$$

3.2.4. Coste total

Sección	Valor (€)
Software	25,00
Mano de obra	14889,00
Hardware	83,33
Total	15112,32

Tabla 15: Desglose de costes del proyecto.

3.3. Gestión de riesgos

Durante la realización de un proyecto se han de tener en cuenta los posibles riesgos que puedan surgir, por lo que es necesario un plan de gestión de riesgos, donde se detalla la prevención y mitigación de los mismos. En esta subsección se detallan los riesgos que se han identificado y cómo se han gestionado.

Baja por enfermedad.
Riesgo: Baja por enfermedad del alumno.
Probabilidad: Baja
Impacto: Alto
Prevención: Se realizará un seguimiento del trabjo con copias de seguridad y una clara documentación.
Mitigación: En caso de enfermedad, se intentará seguir trabajando en la medida de lo posible y se pedirá ayuda a los tutores.

Tabla 16: Baja por enfermedad.

Problemas técnicos.
Descripción: Problemas técnicos con el hardware o software, ya sea un mal funcionamiento o la total incapacidad de uso.
Probabilidad: Media
Impacto: Medio
Prevención: Se realizarán copias de seguridad periódicas y se mantendrán actualizados los sistemas.
Mitigación: En caso de problemas técnicos, se intentará solucionar el problema lo antes posible y se pedirá ayuda a los tutores, se puede continuar trabajando en el otro equipo en caso de que uno falle.

Tabla 17: Problemas técnicos.

Falta de motivación.
Descripción: Falta de motivación para continuar con el proyecto, dejando el trabajo a medias y aplazando lo planeado.
Probabilidad: Baja
Impacto: Muy alto
Prevención: Se realizarán reuniones semanales con los tutores para mantener la motivación y el interés en el proyecto.
Mitigación: En caso de falta de motivación, se intentará buscar nuevas formas de motivación y se pedirá ayuda a los tutores.

Tabla 18: Falta de motivación.

Mala acotación del alcance.
Descripción: Mala acotación del alcance del proyecto durante la planificación, lo que puede llevar a un desbordamiento de las tareas y a un retraso en la entrega.
Probabilidad: Media
Impacto: Alto
Prevención: Se realizarán reuniones semanales con los tutores para revisar el alcance del proyecto y se mantendrá actualizado el plan de trabajo.
Mitigación: En caso de mala acotación del alcance, se intentará redefinir el alcance del proyecto y se pedirá ayuda a los tutores.

Tabla 19: Mala acotación del alcance.

Carencia de datos reales.
Descripción: Carencia de datos reales para la evaluación del modelo, lo que puede llevar a una evaluación poco realista.
Probabilidad: Media
Impacto: Alto
Prevención: Se intentará obtener datos reales para la evaluación del modelo, ya sea mediante la empresa o mediante fuentes externas.
Mitigación: En caso de carencia de datos reales, se intentará simular los datos o se pedirá a la empresa que proporcione datos reales.

Tabla 20: Carencia de datos reales.

Retrasos por dificultad en el desarrollo.
Descripción: Retrasos en el desarrollo del proyecto por dificultades técnicas o de otro tipo.
Probabilidad: Media
Impacto: Medio
Prevención: Se realizarán reuniones semanales con los tutores para revisar el plan de trabajo y se mantendrá actualizado el plan de trabajo.
Mitigación: En caso de retrasos por dificultad en el desarrollo, se intentará redefinir el plan de trabajo y se pedirá ayuda a los tutores, pudiendo repercutir en el alcance del proyecto.

Tabla 21: Retrasos por dificultad en el desarrollo.

Problemas de integración.
Descripción: Problemas de integración de los distintos sistemas que componen el proyecto.
Probabilidad: Baja
Impacto: Alto
Prevención: Se realizarán pruebas de integración periódicas y se mantendrán actualizados los sistemas.
Mitigación: En caso de problemas de integración, se intentará solucionar el problema lo antes posible y se pedirá ayuda a los tutores.

Tabla 22: Problemas de integración.

4. Tecnologías para la gestión del conocimiento

A continuación de detallarán las distintas tecnologías que serán estudiades durante este trabajo de fin de grado. Estas tecnologías representan el estado del arte en la gestión del conocimiento y se van a utilizar para tratar de mejorar la interacción con Jira y la generación de consultas JQL.

4.1. 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.

Los LLMs se basan en arquitecturas de redes neuronales profundas, como los *transformers* [7], 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 [8], Gemini, LLama o Claude, que han demostrado ser capaces de realizar tareas de generación de texto, traducción, resumen, entre otras, con resultados sorprendentes.

Estos modelos de lenguaje tan grandes han sido entrenados con inmensas cantidades de texto, lo que les ha permitido aprender de una gran cantidad de información y generar texto de manera coherente para una gran cantidad de contextos. Sin embargo, estos modelos son muy costosos de entrenar y de mantener, ya que requieren de una gran cantidad de recursos computacionales y de memoria para funcionar correctamente. Además, los datos con los que han sido entrenados no están disponibles en su gran mayoría para el público general, lo que, junto con su coste, ha provocado la popularización de estos como servicios en la nube, en contraposición a una herramienta que se pueda ejecutar en local.

4.2. 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.

4.2.1. 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 Knowledge Graph 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 y que proporcione el mejor contexto para la generación de respuestas. El desarrollo óptimo de esta parte es crucial de cara al rendimiento del sistema, ya que una mala recuperación de información puede llevar a respuestas incorrectas o irrelevantes.

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.

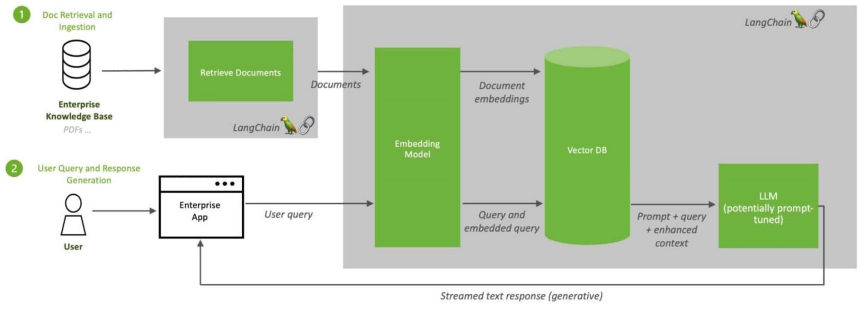

Figura 3: Esquema de funcionamiento de una arquitectura RAG [9].

4.3. 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 números 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 [10] o GloVe [11], 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.

4.4. Ontologías

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 descripción 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 Web Ontology Language (OWL), que es un lenguaje de representación del conocimiento (KR, por sus siglas en inglés) para publicar y compartir ontologías en la web. OWL es desarrollado por el World Wide Web Consortium (W3C)[5] y tanto este como RDFS son lenguajes que proporcionan una forma de referirse a los esquemas de los datos RDF.

4.5. Knowledge Graphs

Los Knowledge Graphs 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, como un grafo tradicional. Los Knowledge Graphs difieren de estos en que las

[5] https://www.w3.org/

entidades están anotadas con términos de una ontología, por lo que respetan la semántica de la misma.

Los Knowledge Graphs 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 Knowledge Graph 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 Knowledge Graph más populares: Resource Description Framework (RDF) y Property Graph Model. El modelo RDF es un estándar abierto desarrollado por W3C[6] para la representación semántica de recursos en la Web. Está formado por declaraciones de la forma sujeto-predicado-objeto, conocidas como tripletas. Por otro lado, los Property Graphs buscan representar la información de manera más expresiva, permitiendo la definición de propiedades en las aristas y los nodos, sin embargo, carecen de la interoperabilidad del modelo RDF por ser desarrollado por W3C.

En el ámbito de la inteligencia artificial, los Knowledge Graphs son especialmente útiles para mejorar el rendimiento de los modelos de lenguaje. Al combinar la información de un knowledge graph con la información de un modelo de lenguaje, se pueden generar respuestas más precisas y acordes al contexto. Ésto 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.

[6] https://www.w3.org/RDF/

5. Diseño

En esta sección se describirá el punto de partida del asistente JiraGPT Next y el 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.

Para estas tres propuestas, se ha hecho un estudio del estado del arte y se han consultado varios artículos que tratan de realizar mejoras similares a las expuestas en este trabajo, logrando inspiración en ellos. Por lo que, se van a probar 3 bases de conocimiento distintas con las que aumentar el contexto que posee el modelo y buscar una mejora en la precisión de las respuestas. Se va a utilizar una ontología para representar las reglas del lenguaje de consulta JQL, una base de datos vectorial con embeddings de la documentación de JQL y un Knowledge Graph que contenga los datos de un proyecto real de LKS Next-GobTech. La figura 4 muestra un diagrama general de las 3 propuestas.

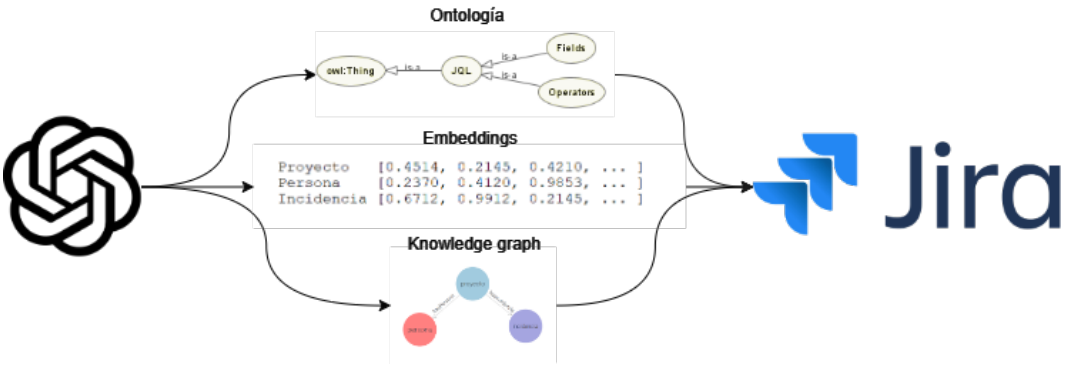

Figura 4: Diagrama de las tres propuestas, simbolizando las 3 bases de conocimiento distintas que van a ser estudiadas.

5.1. Estado inicial

La aplicación desarrollada por Joel García y la presentada en este trabajo difieren en la manera de interactuar con el modelo. La aproximación que tomó Joel García fue la de realizar una ejecución de tres fases distintas, interactuando varias veces con el modelo. En el diagrama de la figura 5, obtenido del trabajo de Joel García [1], muestra el flujo de trabajo de la aplicación JiraGPT Next en su estado inicial.

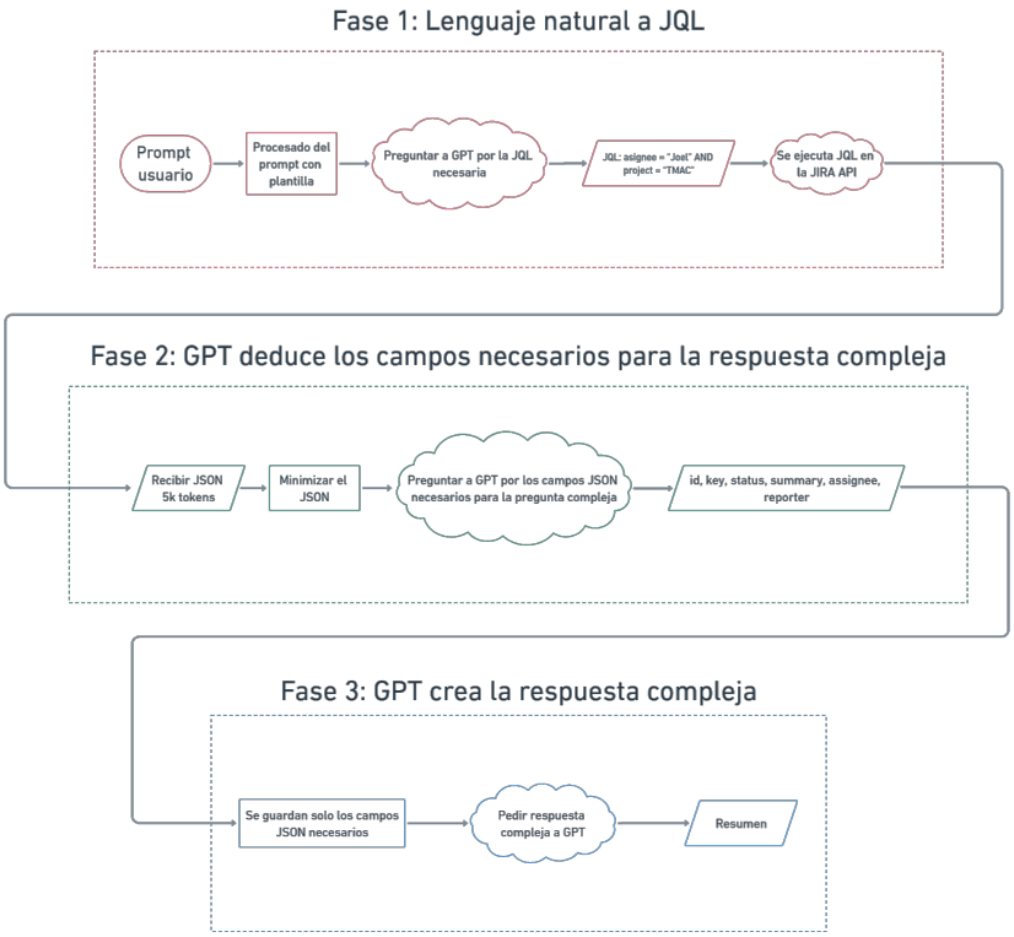

Figura 5: Diagrama de las tres fases de la aplicación JiraGPT Next en su estado inicial [1].

Durante la primera fase se preprocesaba la pregunta para adaptarla con plantillas y otorgar algo de contexto e indicaciones al modelo, se enviaba la pregunta y se postprocesaba la respuesta para limpiar cualquier tipo de comentario adicional que imposibilitase la ejecución de la consulta JQL. Durante esta misma fase se realizaba la llamada a la API de Jira con la consulta JQL generada ya procesada y se recogía el JSON devuelto.

La segunda fase consistía en limpiar el JSON para reducir campos de la respuesta que no contienen información relevante. En caso de no haber activado el botón de pregunta compleja, el resultado sería el JSON limpio, que se expone en la interfaz al usuario. En caso de haber activado la pregunta compleja, se haría una segunda

llamada al modelo para que decidiese cuáles de los campos del JSON que se han conservado son relevantes para responder a la pregunta.

La tercera fase, que se ejecutaba independientemente de si la pregunta compleja ha sido activada, consistía en una tercera llamada para que realizase un resumen de lo preguntado. De esta manera, el usuario recibía tanto los campos que había solicitado como un resumen que respondiese a su pregunta de manera más directa.

Este proceso utilizaba la libería de Python que ofrece OpenAI para comunicarse con el modelo, de manera que resulta necesario adaptar el código para que funcione con esta librería. Lo que se propone desde este trabajo, también es una mejora en la interacción con el modelo, ya que se va a utilizar Langchain, una librería que permite la abstracción del uso del modelo, por lo que es más sencillo realizar llamadas a cualquier modelo de nuestra elección, sin tener que depender de las librerías de cada uno de los proveedores de estos modelos.

5.1.1. Precisión inicial

Para evaluar 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 comprobando 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.

5.2. 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.

5.2.1. Ontología

Durante el inicio de este trabajo se consultaron artículos como *Sequeda et al.* [12], 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 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 [13].

En la figura 6 se muestra una visualización de la ontología y las diferentes jerarquías que existen.

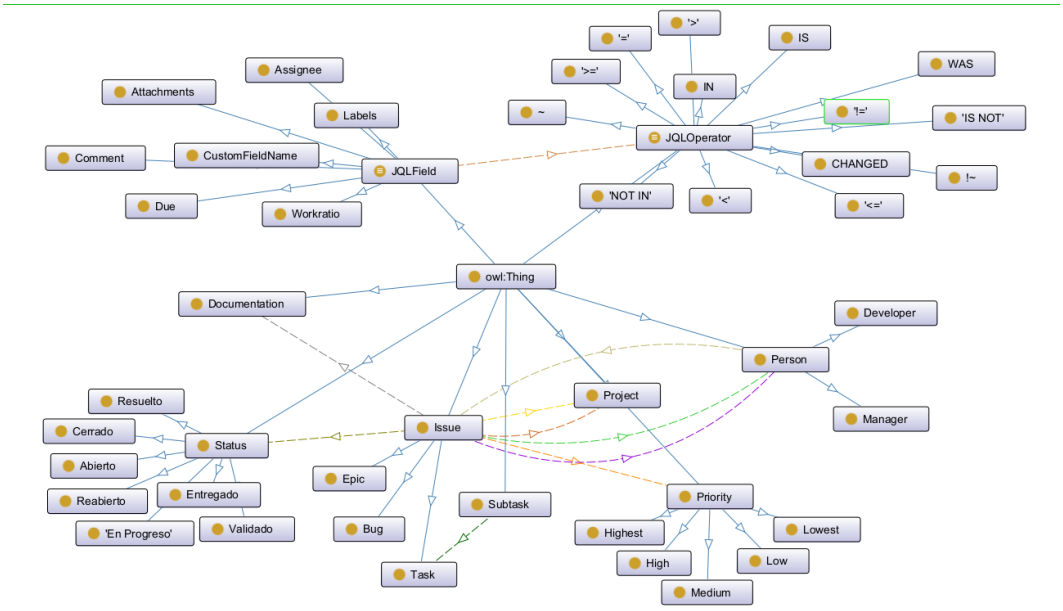

Figura 6: Ontología representativa de JQL y Jira

Se ha decidido hacer la ontología de esta manera para tratar de representar todas las reglas de JQL aprovechando la semántica de una ontología y la estructura de jerarquía de Jira. Entre las distintas clases, que representan campos, operadores, estados y prioridades, también existen reglas que indican si son compatibles en una consulta, sin embargo, al plasmar la ontología en un diagrama como este no se puede apreciar la semántica de las relaciones. Por ejemplo, la siguiente figura muestra las reglas del campo 'Assignee', indicando qué operadores son soportados por este campo, lo cual en la ontología se describe para todos los campos.

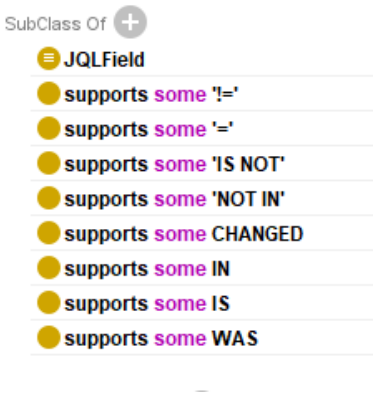

Figura 7: Reglas del campo 'Assignee'

La interacción con la ontología representa un reto, ya que una vez definida la ontología, interactuar con ella de manera que el modelo comprenda lo que se representa es complejo. Se ha optado por utilizar el mismo modelo de lenguaje para buscar los posibles campos relevantes dada una pregunta. Por ejemplo, para la pregunta '¿Qué incidencias tiene asignadas Alberto Aróstegui?', se envía al modelo para que responda indicando qué campos son relevantes para esa pregunta, utilizando una plantilla, adjunta en el anexo I, en la que se le exponen todos los campos que hemos descrito en la ontología, el modelo debería responder con el campo Assignee, que es el campo descrito en la ontología que mejor responde a esa pregunta en su respectiva consulta JQL. Una vez se tienen estos campos, se puede consultar la ontología para obtener toda la información que se represente y que sea relevante para la pregunta, como las subclases que tiene, las restricciones que existen con otros campos u operadores o comentarios que se expongan en la ontología sobre ese campo. El siguiente diagrama muestra cómo es el flujo del sistema.

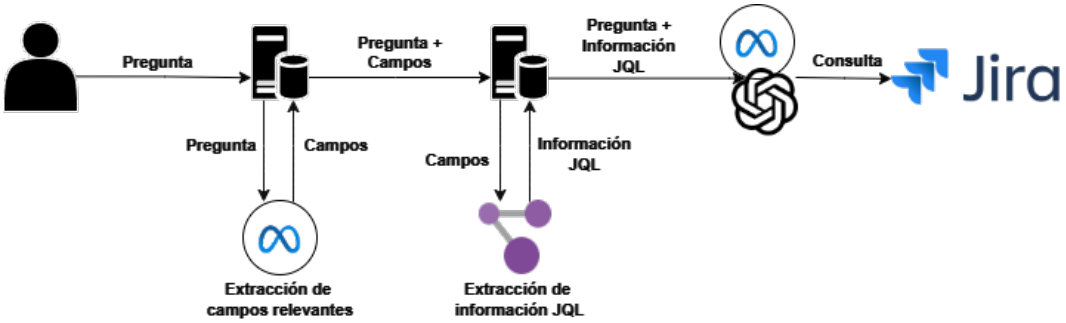

Figura 8: Diagrama de RAG con ontología

5.2.2. Embeddings

De igual manera que en el diseño de la ontología, se propone capturar la documentación oficial de JIRA en esta base de datos vectorial. Esta base de conocimiento almacena 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. Las figuras 9 y 10, extraídas directamente la documentación oficial de Jira, describen un campo JQL y ofrecen ejemplos de uso.

Updated

Search for issues that were last updated on, before, or after a particular date (or date range). Note that if a time-component is not specified, midnight will be assumed. Please note that the search results will be relative to your configured time zone (which is by default the Jira server's time zone).

Use one of the following formats:

`"yyyy/MM/dd HH:mm"`
`"yyyy-MM-dd HH:mm"`
`"yyyy/MM/dd"`
`"yyyy-MM-dd"`

Or use `"w"` (weeks), `"d"` (days), `"h"` (hours) or `"m"` (minutes) to specify a date relative to the current time. The default is `"m"` (minutes). Be sure to use quote-marks (`"`); if you omit the quote-marks, the number you supply will be interpreted as milliseconds after epoch (1970-1-1).

Figura 9: Campo JQL

Figura 10: Ejemplos de uso

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, generando vectores de 1536 dimensiones que representan la información de la documentación de manera numérica. Estos valores de cada dimensión indican una característica de los datos, en caso del texto podría representar conceptos como intenciones o similitudes. De esta manera, oraciones que tengan similitudes, como por ejemplo 'incidencias cuyo estado cambió durante agosto' y, de la documentación, el operador 'changed' que tiene como ejemplo 'Find issues whose status had changed from 'In Progress' back to 'Open'' generarían vectores similares.

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 GPT-4o 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 11 cómo será la interacción entre el modelo y la base de datos vectorial.

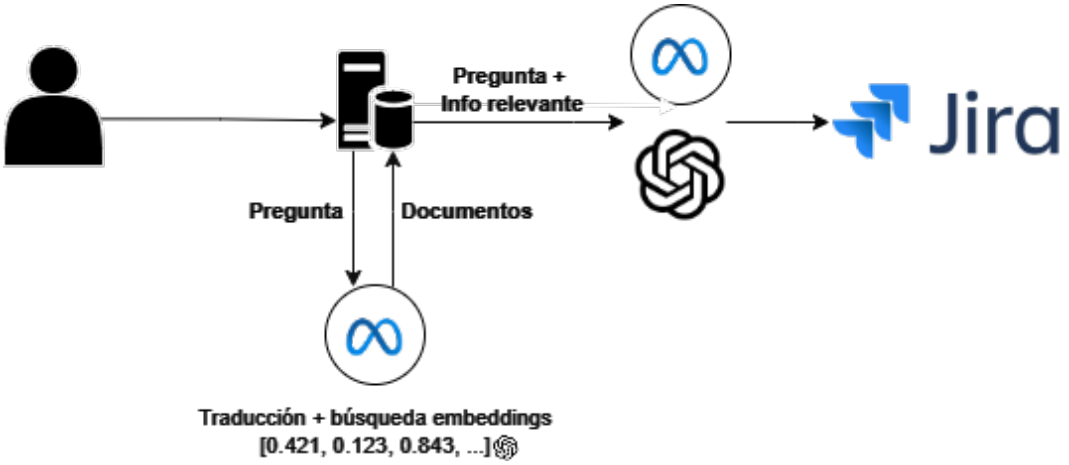

Figura 11: Diagrama de RAG con embeddings

5.2.3. Knowledge Graphs

Como última propuesta para el trabajo, se ha planteado el uso de Knowledge Graphs 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, dada la información de Jira en forma de grafo, el modelo podría ser capaz de inferir cómo es la estructura de información de la plataforma Jira de LKS Next-GobTech y generar respuestas que sean más acordes a lo que se busca. Para esto, se ha de crear un Knowledge Graph que contenga la información de Jira de cada proyecto que tenga activo LKS Next-GobTech, por lo que se ha de crear un programa que sea capaz de transformar esa información en un grafo y se pueda ejecutar de manera periódica, con el fin de que el grafo esté actualizado con la información real que se almacena en Jira. Cada proyecto tendrá su grafo y el programa que crea el grafo a partir de la información de la API debería ser ejecutado periódicamente para mantener la información actualizada.

En la figura 12 se puede apreciar el desglose de una incidencia representada en el grafo para el proyecto ‘MFM‘, con Arkaitz Carbajo como autor (Reporter) y una persona cualquiera asignada a esta tarea. Además, se puede ver que es una tarea de tipo Tarea (No un bug o un error) y es de tipo ‘Issue‘ al ser una incidencia, como todas las demás. Se puede ver que su estado actual es ‘Abierto‘, además, el nodo ‘MFM-35‘ contiene como informacion literal las fechas de creación y vencimiento de esta tarea.

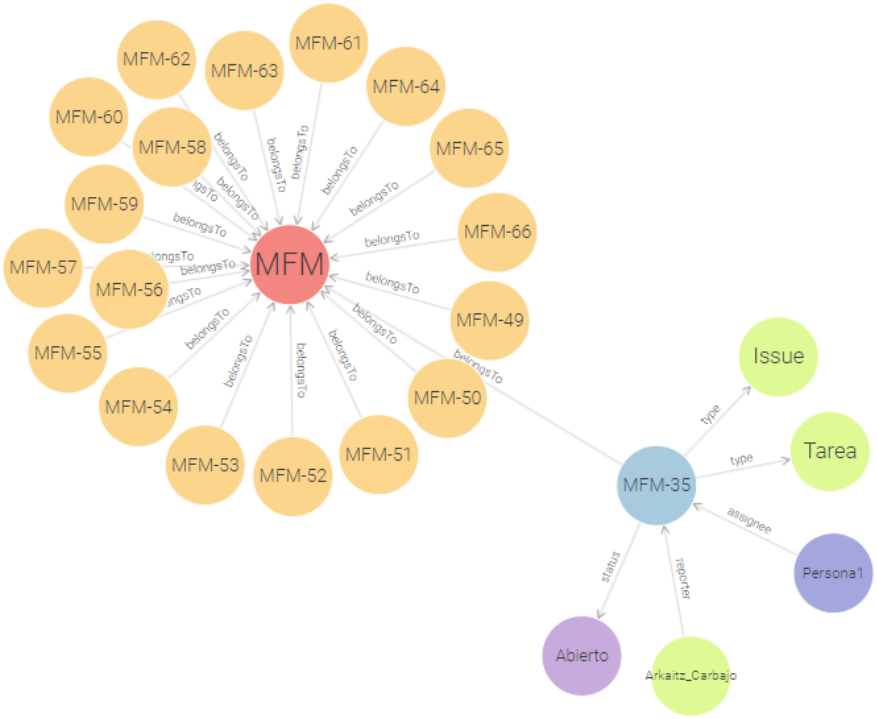

Figura 12: Knowledge Graph para el proyecto MFM, diagrama en GraphDB

El flujo sería el representado en la figura 13, donde se muestra cómo, dada una pregunta de un usuario, se extrae información del knowledge graph que se inyectaría en el prompt para que el modelo pueda generar una respuesta con más contexto.

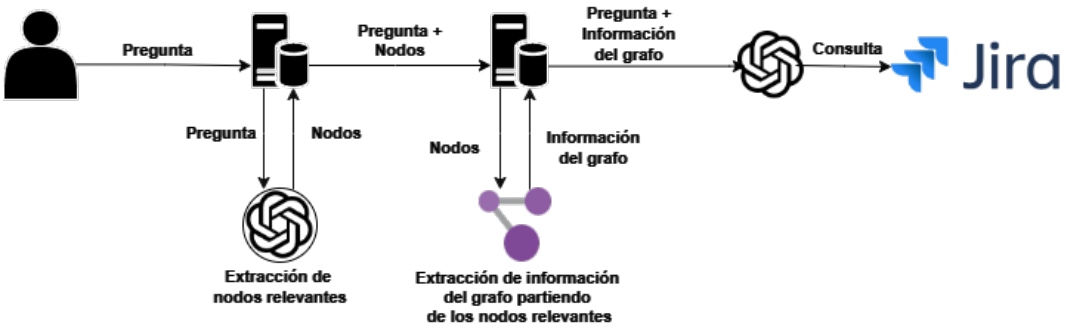

Figura 13: Diagrama de RAG con knowledge graphs

6. Implementación

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

El modelo a utilizar será GPT-4o [14], al cual se harán llamadas a través de programas Python, utilizando la librería Langchain[7] en su versión 0.1.16. En el anexo I se encuentran las distintas plantillas utilizadas para interactuar con el modelo.

6.1. Implementación de RAG: Ontología

Como se ha mencionado en el apartado de diseño, se ha construido una ontología nueva 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 JQL, 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 de JQL, siguiéndo prácticas para describir de manera correcta una ontología, respetando la semántica necesaria.

Esta ontología se ha desarrollado utilizando el software Protégé [15], gratuito y de código abierto, desarrollado por la universidad de Stanford. A través de la librería owlready2[8] se ha podido cargar la ontología en Python y realizar consultas sobre ella.

6.1.1. Interacción con el modelo

La manera más sencilla permitir al modelo interactuar con la ontología es inyectar el archivo entero en prompt, como describe Sequeda et al. [12]. 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, es decir, se van a realizar pruebas con esta aproximación para utilizar como resultado base y evaluar la extracción de información de la ontología.

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.

[7] https://www.langchain.com/
[8] https://pypi.org/project/Owlready2/

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.

Por ejemplo, para la pregunta 'Dame las incidencias en progreso en el proyecto MFM' el modelo devolvería los campos 'status' y 'project' como los más relevantes, y, tras la consulta, tendríamos la siguiente información:

```
['Status', ' Assignee']
Label: Status
Comment: None
Supports operators:
Subclasses of the field:
Abierto
Cerrado
En Progreso
Entregado
Reabierto
Resuelto
Validado


Label: Assignee
Comment: Search for issues that are assigned to a particular user.
You can search by the user's full name, ID, or email address.
EXAMPLES:
Find issues that are assigned to John Smith:
assignee = "John Smith"or
assignee = jsmith
Find issues that are currently assigned, or were previously assigned,
to John Smith:
assignee WAS jsmith
Find issues that are assigned by the user with email address
bob@mycompany.com:
assignee = "bob@mycompany.com"
Supports operators: IS, WAS, !=, IS NOT, =, IN, CHANGED, NOT IN
Subclasses of the field:
```

6.2. 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 [9] 0.5.0, una base de datos vectorial de código abierto que permite realizar búsquedas por similitud de vectores que, además, tiene una sencilla implementación con Langchain.

6.2.1. 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.

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

6.3. Implementación de RAG: Knowledge Graphs

Para crear un grafo de conocimiento de la información de Jira de LKS NextGobTech se ha utilizado la API de Jira para extraer la información de los proyectos activos y se ha escrito un programa en Python que transforma esa información en un grafo. Se ha utilizado la librería RDFlib[10] para la creación del grafo y se ha almacenado en un fichero en formato RDF. Posterior a esto, para realizar consultas al grafo se ha utilizado la librería SPARQLWrapper, que permite realizar consultas SPARQL al grafo RDF.

[9] https://www.trychroma.com/
[10] https://github.com/RDFLib/rdflib

Al igual que en el caso de la ontología, se va a utilizar el modelo de lenguaje para extraer información relevante de la pregunta del usuario, por lo que, dada una pregunta, el modelo devolverá los nodos de los que partir para realizar una consulta SPARQL al grafo. Por ejemplo, dada la pregunta 'Dame las incidencias en progreso en el proyecto MFM ' el grafo devolvería el nodo MFM, desde el que se podría partir y realizar una consulta SPARQL que devolviese los nodos a los que está conectado este, obteniendo, por ejemplo, las incidencias que tiene asignadas o las que reporta.

La consulta SPARQL desde el nodo MFM devolvería, por ejemplo, un resultado similar a este:

```
s: http://vocab.lksnext.com/jira/MFM,
p: http://www.w3.org/1999/02/22-rdf-syntax-ns#type,
o: http://vocab.lksnext.com/jira/Project
```

Con esto, el modelo puede inferir el tipo de estructura que se sigue en el grafo y, por tanto, partir de un mayor contexto para generar las conultas JQL.

7. Ejecución de pruebas

Esta sección describirá las pruebas que se han realizado a las distintas propuestas de mejora del sistema. Primero se detallará el entorno de pruebas en el que han sido ejecutadas, el cual es reproducible, y posteriormente se comentarán los resultados obtenidos.

7.1. Entorno de pruebas

Las pruebas han sido realizadas en un entorno JIRA con varios proyectos: GPT4, que contiene incidencias ficticias y varios proyectos reales de la empresa LKS NextGobTech, que, en el momento de realización de las pruebas siguen en desarrollo y son actualizados diariamente, pero que, por motivos de confidencialidad, no podrán ser revelado. En concreto, se ha realizado sobre 4 proyectos distintos, expuestos en la tabla 23.

Proyecto	**Nº de Incidencias**	**Nº de Personas**
GPT4*	30	2
JIRAGPT*	11	2
MFM	130	7
INGURU	129	5

Tabla 23: Número de incidencias y personas por proyecto.

Los modelos de lenguaje utilizados serán GPT-3.5-turbo, el modelo base de OpenAI, que es barato y rápido y GPT-4o, el más avanzado modelo de OpenAI a fecha de realización del trabajo [14]. Las preguntas utilizadas se encuentran en el anexo I en forma de tabla, con su respectivo JQL esperado.

En cuanto a los resultados, se utilizará el conjunto de 100 preguntas creado durante la realización del trabajo, que contiene una variedad de preguntas de distinta dificultad y con la intención de cubrir la mayor cantidad de casos posibles, se considerará correcta una respuesta si las incidencias devueltas por la API de Jira con la sentencia JQL generada por el modelo son exactamente las mismas que las esperadas, que serán obtenidas mediante otra llamada a la API de Jira con la sentencia JQL predefinida que corresponde a la pregunta.

Los resultados consistirán en una puntuación del 0 al 1, que indicará el número de preguntas respondidas de manera correcta.

7.2. Resultados

Los resultados obtenidos, evaluados tanto para GPT-3.5-turbo como para GPT-4o se muestran a continuación, en las figuras 14 y 15, respectivamente. Se dividen en dos gráficas distintas, las cuales muestran el estado inicial, que es la ejecución sin ningún sistema de RAG y las tres propuestas de mejora:

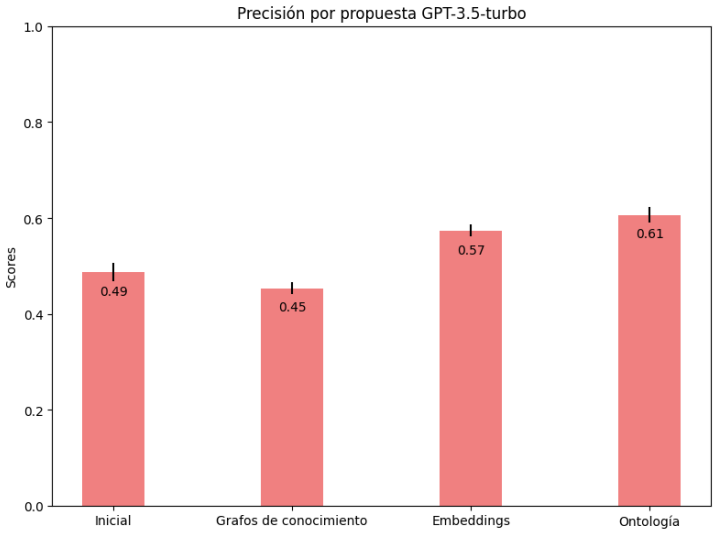

Figura 14: Resultados para GPT-3.5-turbo

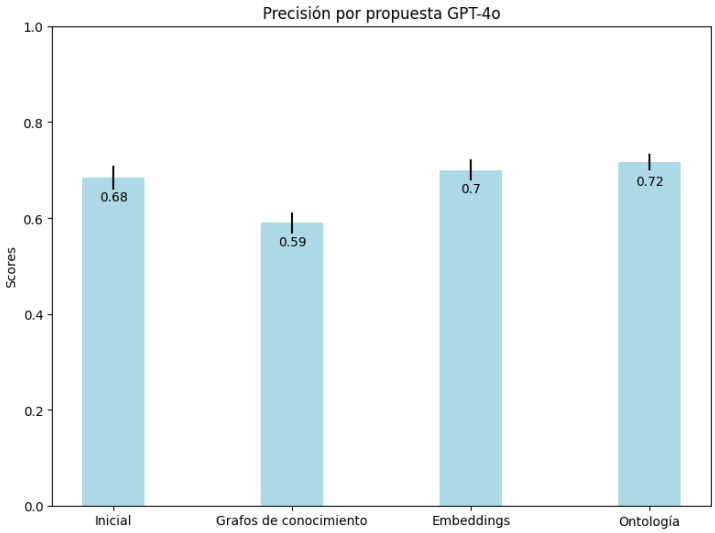

Figura 15: Resultados para GPT-4o

En las tablas se puede apreciar como la mejora en el modelo GPT-3.5 es mayor que en el modelo GPT-4o. Partiendo de la precisión inicial de 0.49 de GPT-3.5 sin técnicas de RAG, la arquitectura utilizada en el trabajo de Joel García, se incrementa hasta 0.57 con la propuesta de embeddings y 0.61 con la propuesta de la ontología, esto supone un aumento de alrededor de 10 puntos porcentuales si se usa la media de ambos. En el caso de GPT-4o, la precisión inicial es de 0.68, y se incrementa hasta 0.7 con la propuesta de embeddings y 0.72 con la propuesta de la ontología, lo que supone un aumento de 3 puntos porcentuales si se toma la media como medida. En ambos casos, la propuesta de la ontología resulta ser la que mejores resultados genera, aunque en el caso de GPT-4o, la mejora es menor que en el caso de GPT-3.5.

En cuanto a la mejora individual de cada una de las propuestas, tanto la opción de la ontología como la de los embeddings supone una mejora en precisión, mientras que la propuesta de los KGs reduce la precisión. Esto puede deberse a que, tanto la ontología como los embeddings, describen un concepto como JQL y aportan contexto sobre cómo crear este tipo de consultas, mientras que la opción de los KGs aporta datos sobre las incidencias de Jira, lo que requeriría que el modelo de lenguaje infiriese cómo funcionan las consultas JQL o la estructura de los proyectos de la empresa para poder generar consultas que sean acordes a ello, lo cual puede ser más complicado.

La diferencia entre la ontología y los embeddings es la manera en la que se representa la información de JQL, mientras que en la ontología se representa como las reglas haciendo uso de la representación semántica de una ontología, en los embeddings se representa como una serie de ejemplos de la documentación que se tratan de relacionar con la pregunta del usuario. La ontología por ende, parece ser una representación más adecuada para el problema, ya que la semántica de las reglas de JQL se puede llegar a expandir con una mayor cantidad de información y se puede explotar aún más la semántica de la ontología para mejorar la generación de consultas JQL.

Si miramos el grano fino, hay preguntas que el modelo nunca hace correctamente si no es aportado contexto, como por ejemplo ‘Muestra las incidencias que se han pasado de horas y por cuánto ‘. Esta pregunta el modelo la resuelve con el JQL ’timespent > originalestimate’, lo cual, si no existe un campo ’originalestimate’, no da resultados. Sin embargo, haciendo uso de la ontología, suele responder con ’workratio > 100’, que es una respuesta que da resultados en cualquier caso. En el caso de los embeddings y el KG, no suele ser capaz de responderla de manera correcta tampoco.

El tipo de consultas que más complejas resultan para el modelo suelen ser las que

esperan JQL que hace uso de 'CHANGED TO <Estado> DURING', ya que el modelo no parece conocer este tipo de uso. Al tener LKS Next-GobTech una serie de estados para las incidencias que son definidos por ellos mismos, como 'Aprobada', 'En Progreso', 'Validada' etcétera, hay preguntas que requieren hacer uso de este tipo de consulta JQL, como por ejemplo '¿Qué incidencias fueron resueltas durante mayo de 2024?', que necesitaría una consulta JQL como 'status CHANGED TO Resuelta DURING (2024-05-01,2024-05-31)' ya que lo que se busca son las incidencias que han cambiado al estado 'Resuelta', pero el modelo suele responder con 'status = 'Resuelto' AND resolved >= '2023-08-01' AND resolved <= '2023-08-31', que no devuelve las mismas incidencias. Esto con el uso de las ontologías parece mejorar, pero no de manera significativa, a veces usa este tipo de consultas, pero sigue comentiendo errores. De cara a una mejora del sistema, se podría buscar una manera de que el modelo aprenda este tipo de consultas, ya que parece ser un punto débil en la generación de consultas JQL.

Otro tipo de incidencias que el modelo falla la mayoría de intentos cuando se ejecuta sin contexto son las que piden incidencias con una prioridad concreta, por ejemplo, si se piden las incidencias de máxima prioridad, el modelo suele responder con 'priority = 'Máxima', que devuelve un error, ya que la prioridad es un campo fijo de Jira que utiliza 5 valores en inglés: 'Lowest', 'Low', 'Medium', 'High' y 'Highest', al ser preguntado en español y los campos ser palabras en inglés, parece confundirlo. Cuando se aporta contexto, principalmente la ontología, ya que se definen explícitamente los niveles de prioridad en esta, suele cometer menos fallos, pero no siempre acierta, a veces no parece querer aprovechar el contexto proporcionado por la ontología.

En el caso de grafo no se expone explícitamente ningún operador ni campo JQL, por lo que preguntas que el modelo base falla por no utilizar un campo concreto, como 'workratio', las falla también. El modelo además, parece confundir más los campos y operadores y no parece ser capaz de inferir cómo es la estructura de la información de Jira de LKS Next-GobTech.

8. Conclusiones

8.1. Objetivos

El objetivo de este trabajo, desde el primer momento, ha sido estudiar cómo se puede implementar un sistema de RAG para otorgar contexto a un modelo, con la intención de mejorar la precisión con la que genere consultas JQL, ya que los modelos de lenguaje actuales, no parecen poseer mucho contexto sobre la generación de consultas JQL, además, si no son conocedores de la estructura de los proyectos, no pueden generar ciertas consultas complejas que sean útiles para los usuarios.

Los objetivos logrados son acordes con los establecidos al principio del trabajo, ya que lo que se buscaba era realizar un estudio y comprobar la potencial mejora de los modelos. En vistas de lo logrado, 10 puntos porcentuales de mejora en GPT-3.5-turbo y 3 puntos porcentuales en GPT-4o, se puede considerar que existe una mejora de precisión con la implementación de un sistema de RAG. Sin embargo, queda visto que la mejora en modelos más avanzados no es tan significativa como la que se logra con modelos más básicos. Desde un punto de vista comercial, ejecutar modelos más básicos resulta más barato y rápido, por lo que la implementación de un sistema de RAG puede ser más deseable en este caso, ya que un sistema implementado con un modelo básico y técnicas de RAG se acerca a la precisión lograda por un sistema con un modelo más avanzado.

8.2. Conclusiones

Cabe recalcar que varias de estas distintas tecnologías propuestas, como los Knowledge Graphs (KGs) o las ontologías, han requerido de un estudio previo para poder ser implementadas en el proyecto.

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.

La parte de los grafos de conocimiento supone un reto, ya que es uno de los ejes del trabajo que más dudas ha generado por no ser una representación de un concepto, sino una representación de los datos de la empresa en Jira directamente, lo cual significa que el modelo tiene que inferir cómo funciona JQL o la estructura de proyectos de la empresa para poder generar consultas que sean acordes a ello.

8.3. Conclusiones personales

Este trabajo, aunque ha tenido sus altibajos, ha sido una experiencia que calificaría como altamente positiva. En un principio, la idea que se me planteó de adentrarme en un mundo que no conocía, como es el de los grafos de conocimiento o las ontologías, me pareció bastante llamativa por el simple hecho de que iba a aprender algo nuevo, además, juntándolo con el estado del arte de los modelos de lenguaje, parecía que iba a ser realmente enriquecedor, y así ha sido. Ha habido momentos de mayor frustración, donde parecía que la implementación de las diferentes propuestas estaba un poco 'cogida con pinzas', pero la investigación es así, y es lo que la hace interesante. Si algo saco en claro de este trabajo, es que la investigación es un camino que no es sencillo, pero es muy satisfactorio y no podría estar más agradecido de haber hecho el trabajo con Arkaitz, Mikel y Unai, que me han ayudado en todo momento y que me han hecho apreciar lo que estaba haciendo.

8.4. Trabajo futuro

En cuanto al trabajo posible realizable en este proyecto, se podrían proponer distintas mejoras, como la creación de un grafo más complejo y con una mejor representación de la información o una ontología que cubriese de mejor manera los conceptos enteros de JIRA, no solo JQL, ya que este no parece contener tanta semántica como para que una ontología parezca realmente útil. Mencionar que, durante el transcurso del trabajo, se pidió un proyecto de Hazitek para obtener financiación y desarrollar el proyecto más allá de su estado actual, con una posible mejora subsecuente.

9. Anexo I

9.1. Conjunto de preguntas

Las 100 preguntas utilizadas para la evaluación de las distintas propuestas de mejora y la consulta JQL que se utiliza para comprobar las incidencias esperadas.

ID	Pregunta	Consulta JQL
1	Muestra las incidencias en progreso	status = 'En progreso'
2	Muestra las incidencias en progreso y sus horas acumuladas	status = 'En progreso'
3	Muestra las incidencias que ya estan entregadas a cliente	status = 'Entregado'
4	Muestra las incidencias que se han pasado de horas y por cuánto	workratio > 100
5	Muestra las incidencias que deberían acabar en agosto de 2023	due >= '2023-08-01' AND due <= '2023-08-31'
6	Muestra las incidencias sin cerrar ni resolver	status not in (Resuelto, Cerrado)
7	Muestra quién tiene menos carga de trabajo	ORDER BY assignee ASC, updated DESC
8	Muestra quién tiene menos carga de trabajo en MFM	project = 'MFM'
9	Cuántos tickets se han resuelto en agosto de 2023?	status changed to Resuelto DURING (´2023-08-01´, ´2023-08-31´)
10	Cuántos tickets se han resuelto en agosto de 2023?	status changed to Resuelto DURING (´2023-08-01´, ´2023-08-31´)
11	Muéstrame todas las incidencias abiertas asignadas a Alberto Aróstegui García	assignee = ´Alberto Aróstegui García´ AND status = Abierto
12	Muéstrame todas las incidencias con estado Abierto asignadas a Alberto Aróstegui García	assignee = ´Alberto Aróstegui García´ AND status = Abierto
13	¿Cuántos tickets sin resolver hay?	status != Resuelto
14	Encuentra las incidencias asignadas a Alberto Aróstegui García.	assignee = ´Alberto Aróstegui García´
15	¿Cuántas incidencias se cerraron en agosto de 2023?	status changed to Cerrado DURING (´2023-08-01´, ´2023-08-31´)

Continúa en la siguiente página

ID	Pregunta	Consulta JQL
16	¿ que incidencias fueron cerradas durante agosto de 2023?	status changed to Cerrado DURING (´2023-08-01´, ´2023-08-31´)
17	Muestra todas las incidencias con alta prioridad	priority = High
18	Dame una lista de las incidencias cuyo estado cambio en agosto de 2023 .	status CHANGED DURING (´2023-08-01´, ´2023-08-31´)
19	Encuentra todos los tickets con ´base de datos´ en el resumen .	summary ´base de datos´
20	¿Cuántos tickets están resueltos ?	status = Resuelto
21	Muestra todas las incidencias reportadas por Alberto Aróstegui García	reporter = ´Alberto Aróstegui García´
22	Muestra el progreso de la incidencia con el campo personalizado Custom_Id con valor GPT4-30	Custom_Id GPT4-30
23	¿Cuáles son las incidencias creadas en agosto de 2023 ?	created >= '2023-08-01' AND created <= 2023-08-31
24	¿Cuáles son las incidencias creadas en agosto de 2023 ? Ordenalas de mayor a menor prioridad	created >= '2023-08-01' AND created <= '2023-08-31' ORDER BY priority DESC
25	¿Cuántos bugs se han reportado?	issuetype = Bug
26	Dame todas las incidencias en la fase de ´Resuelto´.	status = 'Resuelto'
27	Muéstrame todas las incidencias reabiertas	status = Reabierto
28	Muéstrame todas las incidencias con estado Reabierto'	status = Reabierto
29	Muestra todos los tickets que vencen en agosto de 2023 .	due >= '2023-08-01' AND due <= '2023-08-31'
30	¿Qué tickets terminan después del 1 de septiembre de 2023 ?	due > '2023-09-01'
31	Encuentra todas las incidencias movidas a ´Validado´ en agosto de 2023	status changed to Validado DURING (´2023-08-01´, ´2023-08-31´)
32	Dame las incidencias resueltas en agosto de 2023	status changed to Resuelto DURING (´2023-08-01´, ´2023-08-31´)
33	¿Cuántos tickets están en fase de ´En Progreso´ ?	status = ´En progreso´

Continúa en la siguiente página

ID	Pregunta	Consulta JQL
34	Lista todos los tickets creados en agosto de 2023	created >= '2023-08-01' AND created <= '2023-08-31'
35	Muéstrame las incidencias abiertas más antiguas .	status = Abierto and ORDER BY created ASC
36	Muéstrame las incidencias con estado Abierto' más antiguas .'	status = 'Abierto' ORDER BY created ASC
37	Lista todos los tickets con la etiqueta ´seguridad´ y dame un resumen .	labels = 'seguridad'
38	¿Cuántas incidencias se han movido a ´En progreso´ en agosto de 2023 ?	AND status changed to ´En progreso´ DURING (´2023-08-01´, ´2023-08-31´)
39	¿Cuáles son las incidencias de máxima prioridad asignadas a Alberto Aróstegui García ?	AND assignee = 'Alberto Aróstegui García' AND priority = Highest
40	Lista todas las incidencias cerradas en agosto de 2023	AND status changed to Cerrado DURING (´2023-08-01´, ´2023-08-31´)
41	¿Cuántas incidencias nuevas se han agregado en agosto de 2023?	AND created >= '2023-08-01' AND created <= '2023-08-31'
42	Encuentra todas las incidencias con ´rendimiento´ en su descripción .	description rendimiento
43	Encuentra todos los tickets movidos a ´Cerrado´ en agosto de 2023 .	status changed to Cerrado DURING (´2023-08-01´, ´2023-08-31´)
44	Muestra todas las incidencias que están vencidas	due < now() and status != Cerrado
45	Lista todos los tickets en los que alguien ha votado .	voter is not empty
46	¿Cuántas incidencias están en la fase ´Abierto´ ?	AND status = Abierto
47	¿Cuántas incidencias están abiertas ?	AND status = Abierto
48	Recupera las incidencias que tengan que ver con Gitlab	summary 'Gitlab' OR description 'Gitlab'
49	Muestra las incidencias sin asignar	AND assignee is EMPTY
50	Cuantas incidencias asignadas hay ?	AND assignee is not EMPTY
51	Muestrame las incidencias abiertas	status not in (Resuelto, Cerrado)
52	Que incidencias de maxima prioridad	and priority = Highest

Continúa en la siguiente página

ID	Pregunta	Consulta JQL
53	Selecciona las incidencias sin resolver de esta semana	created >= startOfWeek() and status != Resuelto
54	Muestra todas las incidencias con enlaces externos	attachments is not EMPTY
55	Selecciona las incidencias que vencen en los proximos 3 dias	due <= endOfDay('+3d') and status != Resuelto
56	Muestra las incidencias en las que se esta trabajando ahora mismo	status = 'En Progreso'
57	Muestra las incidencias creadas esta semana	and created >= startOfWeek()
58	Muestra las incidencias creadas en los ultimos 7 dias	created >= -7d()
59	Muestrame las incidencias que vencen el 11 de agosto de 2023	due = '2023-08-11'
60	Dame las incidencias que contengan etiquetas	labels is not EMPTY
61	Muestrame las incidencias asignadas a Alberto Aróstegui García que tengan etiquetas	assignee = 'Alberto Aróstegui García' and labels is not EMPTY
62	Dame las incidencias asignadas a mi que contengan enlaces externos	assignee = currentUser() and attachments is not EMPTY
63	Dame las incidencias asignadas a mi	assignee = currentUser()
64	Muestrame las incidencias que estan en un sprint abierto	sprint in openSprints()
65	Que incidencias estan en un sprint ya cerrado	sprint in closedSprints()
66	¿Cuáles son los ítems que tienen una fecha de vencimiento entre el 1 de agosto de 2023 y el 31 de agosto de 2023 ?	due >= '2023-08-01' AND due <= '2023-08-31'
67	¿En que proyectos ha votado al menos una persona?	voter is not EMPTY
68	¿Qué inicidencias tengo yo asignadas ?	assignee = currentUser()
69	¿Qué incidencias están resueltas ahora mismo ?	status = Resuelto
70	¿Qué tickets se cerraron durante agosto de 2023?	status changed to Cerrado DURING (´2023-08-01´, ´2023-08-31´)
71	¿Qué tickets cambiaron de estado durante agosto de 2023?	status changed DURING (´2023-08-01´, ´2023-08-31´)
72	¿Qué incidencias fueron creadas el último mes ?	created >= startOfMonth(-1)

Continúa en la siguiente página

ID	Pregunta	Consulta JQL
73	¿Cuántas incidencias están cerradas pero no resueltas?	status = Cerrado and status != Resuelto
74	Muestrame las incidencias que se han movido a Resuelto en agosto de 2023	status changed to Resuelto DURING (´2023-08-01´, ´2023-08-31´)
75	Dame las incidencias que han sido actualizadas en las ultimas 24 horas	updated >= -24h()
76	¿Que incidencias vencen hoy ?	due <= endOfDay()
77	¿Cuantas incidencias estan abiertas pero no asignadas?	status = Abierto and assignee is EMPTY
78	¿Qué incidencias fueron resueltas en los últimos 30 días ?	status changed to Resuelto DURING (´-30d´, now())
79	¿Cuántas incidencias se han movido a ´En progreso´ en los últimos 7 días ?	status changed to ´En progreso´ DURING (´-7d´, now())
80	¿Qué incidencias están en un sprint y no están asignadas?	sprint in openSprints() and assignee is EMPTY
81	¿Que incidencias tienen enlaces externos?	attachments is not EMPTY
82	Lista todas las incidencias que contengan la palabra ´urgente´	summary ´urgente´
83	Dame las incidencias de prioridad alta que todavia no estan asigndas	priority = High and assignee is EMPTY
84	Muestramen las incidencias cerradas el último trimestre	status changed to Cerrado DURING (´-3M´, now())
85	Dame las incidencias de baja prioridad que estan en progreso	priority = Low and status = ´En progreso´
86	¿Qué incidencias tienen un campo personalizado con valor ´GPT4-30´?	Custom_Id GPT4-30
87	¿Cuántas incidencias se han creado en los últimos 7 días ?	created >= -7d()
88	Muestra las incidenciasque están marcadas como críticas y aún están abiertas	priority = Highest and status = Abierto
89	Dame las incidencias que han cambiado de prioridad	priority CHANGED
90	¿Cuántas incidencias relacionadas con fallos hay ?	issuetype = Bug
91	Dame el total de horas registradas en las incidencias tipo Epic	issuetype = Epic

Continúa en la siguiente página

ID	Pregunta	Consulta JQL
92	¿Qué incidencias están Resueltas pero aun no se han cerrado?	status = Resuelto and status != Cerrado
93	¿Cuántas incidencias se han creado en el último mes ?	created >= startOfMonth(-1)
94	Dame las incidencias que han sido actualizadas en las últimas 24 horas	updated >= -24h()
95	¿Qué incidencias están ahora mismo validadas ?	status = Validado
96	¿Cuántas incidencias se han movido a ´En progreso´ en los últimos 7 días ?	status changed to ´En progreso´ DURING (´-7d´, now())
97	¿Qué incidencias están en un sprint y no están asignadas?	sprint in openSprints() and assignee is EMPTY
98	¿Qué incidencias de baja prioridad están en progreso ?	priority = Low and status = ´En progreso´
99	Dame las incidencias que han cambiado de prioridad	priority CHANGED
100	¿Cuántas incidencias relacionadas con fallos hay ?	issuetype = Bug

9.2. Plantillas para la interacción con el modelo

```
template_traduccion = '''<|begin_of_text|><|start_header_id|>system
<|end_header_id|>
You are an expert translator for JIRA issues.
Your task will be to translate the following question, which asks for
JIRA issues from Spanish to English. You only have to return the
translation, do not return any additional comments. So, for example,
for the question '¿Qué incidencias estan En Progreso?' you should
return 'What issues are In Progress'.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_rag_ontologia = '''<|begin_of_text|><|start_header_id|>system
<|end_header_id|>
You are a JQL assistant for question-answering tasks.
Use the following ontology to answer the questions, which contains
information about JIRA issues and how to generate JQL queries. You
```

```
only have to return the JQL query for it to run verbatim, do not
return any additional comments. So, for example,for the question
'Que incidencias estan En Progreso?' you should return 'status =
'En Progreso'. Utiliza siempre los nombres de estado en castellano,
entonces 'In Progress' sera 'En Progreso''.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: {context}
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_rag_embeddings = '''<|begin_of_text|><|start_header_id|>
system<|end_header_id|>
You are a JQL assistant for question-answering tasks.
Use the following documents as support, which are examples that come
from the official Atlassian documentation to answer the questions.
You only have to return the JQL query for it to run verbatim,
do not return any additional comments. So, for example, for the
question 'Que incidencias estan En Progreso?' you should return
'status = 'En Progreso'. Always use the status names in Spanish,
so 'In Progress' will be 'En Progreso'.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: {context}
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_retrieve_ontology_fields = '''<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>
You are a JQL assistant for question-answering tasks.
Given the following JQL fields in the context and a question,
you must answer with a list of the relevant fields for the
given question, return the fields separated by commas.
For example, for the question '¿Cuándo vencen las incidencias asignadas
a Alberto Arostegui?' you should return 'Assignee, Due'. Do not return
any additional comments. Return ONLY 2 fields at most.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: Assignee, Attachments, Comment, CustomFieldName, Due, Labels,
Workratio, Status, Priority
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_ontology_fields = '''<|begin_of_text|><|start_header_id|>
system<|end_header_id|>
You are a JQL assistant for question-answering tasks.
```

```
Given the following JQL fields and some additional information based
on an ontology describing how to generate JQL queries and a question,
you must answer with the JQL query that will answer the given
question. You only have to return the JQL query for it to run verbatim,
do not return any additional comments. So, for example, for the question
'Que incidencias estan En Progreso?' you should return 'status = 'En
Progreso'.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: {context}
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_retrieve_graph_nodes = '''<|begin_of_text|><|start_header_id|>
system<|end_header_id|>
You are a JQL assistant for question-answering tasks.
Given the following question, you must answer with a list of the
relevant nodes for a conceptual JIRA graph, return the fields
separated by commas. So, you would return relevant fields for
the given question, the project, the person, the issue. For
example, for the question '¿Cuándo vencen las incidencias de MFM
asignadas a Alberto Arostegui?' you should return 'MFM, Alberto
Arostegui, Issue, assignee'. Do not return any additional comments.
In the context there are possible nodes.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: Person, Issue, assignee, Project, due
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''

template_rag_graph = '''<|begin_of_text|><|start_header_id|>
system<|end_header_id|>
You are a JQL assistant for question-answering tasks.
Use the following documents as support, which are examples that
come from a Knowledge Graph made from the Jira data of a project.
You only have to return the JQL query for it to run verbatim,
do not return any additional comments. So, for example, for
the question 'Que incidencias estan En Progreso?' you should
return 'status = 'En Progreso'. Always use the status names in
Spanish, so 'In Progress' will be 'En Progreso'.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Context: {context}
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''
```

```
template_standalone = '''<|begin_of_text|><|start_header_id|>
system<|end_header_id|>
You are an AI assistant trained on JIRA Query Language. Your
task is to fullfil users questions.\nAlways use the following
issue status names in Spanish and never in English: "Open"
should be "Abierto", "In Progress" should be "En Progreso",
"Resolved" should be "Resuelto", "Approved" should be "Aprobada",
"Delivered" should be "Entregado", "Reopened" should be
"Reabierto", "Closed" should be "Cerrado". You should return
the jql query verbatim without any additional comments.
For example, for the question 'Que incidencias estan En Progreso?'
you should return 'status = 'En Progreso''.
<|eot_id|><|start_header_id|>user<|end_header_id|>
Question: {question}
Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>'''
```

Referencias

[1] Joel Moisés García Escribano. «JiraGPT Next:asistente basado en IA para gestión de proyectos». En: (2023). URL: https://addi.ehu.es/handle/10810/68412.

[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] Glassdoor. *Glassdoor*. Accessed: 2024-03-30. URL: https://www.glassdoor.es/Sueldos/it-project-manager-sueldo-SRCH_KO0,18.htm.

[7] 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.

[8] 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/%20file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[9] Rick Merritt. *What Is Retrieval-Augmented Generation, aka RAG?* Accessed: 2024-03-30. URL: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/.

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

[11] 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.

[12] 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].

[13] 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/.

[14] OpenAI. *GPT-4o*. Accessed: 2024-05-13. URL: https://openai.com/index/hello-gpt-4o/.

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