



**FACULTAD DE INGENIERÍA, ARQUITECTURA Y DISEÑO
MAGISTER EN DATA SCIENCE**

**INTEGRACIÓN DE NLP EN LA GESTIÓN DE TICKETS DE
SOPORTE TÉCNICO**

Proyecto para optar al Grado de
Magister en Data Science

Profesor Guía : Matías Greco

Profesor Co-Guía :

Estudiante : Harold Gómez

© xxxxxxxxxxxx

Queda prohibida la reproducción parcial o total de esta obra en cualquier forma,
medio o procedimiento sin permiso por escrito de los autores.

Santiago de Chile
2025

HOJA DE CALIFICACIÓN

En Santiago, el _____ de 2025, los abajo firmantes dejan constancia que el estudiante [NOMBRE] del Magíster en Data Science ha aprobado la tesis/proyecto para optar al Grado de Magíster en Data Science con una nota de

(Nombre y firma profesor evaluador)

(Nombre y firma profesor evaluador)

DEDICATORIA

Dedico este proyecto con profundo amor y gratitud a mi esposa Andrea, compañera de vida, quien con paciencia infinita, cariño constante y apoyo incondicional fue mi refugio y fortaleza en cada paso dado durante estos años de esfuerzo.

A mis queridos hijos, Nicolás y Martina, fuente inagotable de alegría, ternura y energía; gracias por ser la luz que ilumina mi camino y la razón más profunda detrás de cada meta que emprendo.

A ustedes, con todo mi corazón.

AGRADECIMIENTOS

Quiero expresar mi más profunda gratitud a mis padres, Luis y Carmen, quienes con sabiduría y amor formaron los cimientos que sostienen cada uno de mis logros. A mi padre, Luis, quien con su ejemplo me enseñó desde niño que el deseo de triunfar es la mitad de la victoria; y a mi madre, Carmen, quien me mostró con ternura infinita lo que realmente significa el amor incondicional.

Agradezco de corazón a mis hermanos, Ronny y Michael, cuyo apoyo constante, complicidad y cercanía fueron siempre un estímulo invaluable durante estos años de esfuerzo y estudio.

Mi especial reconocimiento a mi profesor guía, Matías Greco, por su dedicación generosa, valiosas sugerencias y paciencia durante todo el proceso de elaboración de este proyecto, permitiéndome convertir los desafíos en oportunidades de crecimiento.

Finalmente, agradezco a mis compañeros de grupo, Jesús Rodríguez y Gonzalo Cayunao, por haber sido excelentes compañeros en este camino compartido, por la calidad de sus aportes y, sobre todo, por hacer de esta experiencia académica una etapa llena de aprendizajes, compañerismo y colaboración genuina.

A todos ustedes, gracias por haber estado presentes en este importante logro de mi vida.

RESUMEN

El presente proyecto aborda el desafío de asistir en la resolución de tickets de soporte técnico mediante la reutilización automatizada de conocimiento existente, específicamente documentación oficial y foros públicos. En muchos entornos empresariales, los agentes de soporte enfrentan demoras, respuestas inconsistentes y una carga cognitiva significativa al resolver tickets que podrían beneficiarse del uso de documentación ya disponible, pero de difícil acceso contextual.

A nivel del estado del arte, existen soluciones empresariales que aplican modelos de lenguaje y clasificación automática de tickets, como los ofrecidos por Microsoft, Salesforce y ServiceNow. Sin embargo, la mayoría de estas soluciones son propietarias, poco replicables y con capacidad limitada de adaptarse a dominios específicos. Este proyecto se diferencia por construir una solución abierta y trazable que aplica recuperación semántica mediante embeddings personalizados y motores vectoriales como Weaviate, con un enfoque en transparencia, reproducibilidad y adaptabilidad al dominio técnico de Azure.

Para alcanzar este objetivo, se diseñó una arquitectura compuesta por tres bloques principales: (1) extracción automatizada de datos públicos desde Microsoft Learn y Microsoft Q&A mediante técnicas de scraping; (2) generación de embeddings utilizando modelos preentrenados (S-BERT y E5) y comparación con embeddings generados por OpenAI; y (3) almacenamiento y consulta semántica en una base de datos vectorial con esquemas personalizados. Esta infraestructura permite comparar estrategias de vectorización y evaluar la calidad de recuperación frente a consultas simuladas, sentando las bases para una futura implementación en contextos reales de soporte técnico.

TODO: AGregar Resultados y Conclusiones

Palabras clave: soporte técnico, procesamiento de lenguaje natural, recuperación semántica, embeddings, Weaviate, Microsoft Azure.

ABSTRACT

This research paper proposes an automated solution to improve the management of technical support tickets through natural language processing (NLP) and semantic information retrieval. Many companies face challenges when responding to tickets without tools that support document consultation, resulting in delays, inconsistent responses, and a significant operational burden. Using public data from Microsoft Azure, this project develops a system capable of associating frequently asked questions with technical articles using a vector database, thereby facilitating access to relevant information in real-world support contexts.

To achieve this objective, data were collected and processed using scraping techniques. The texts were then vectorized using pre-trained language models, and a vector database with semantic search capabilities was integrated. Subsequently, the retrieval accuracy of the system was evaluated, demonstrating that it can be an effective tool to reduce ticket resolution time.

Keywords: technical support, NLP, semantic retrieval, vector database, Microsoft Azure

CONTENIDOS

DEDICATORIA	4
AGRADECIMIENTOS	5
RESUMEN	6
ABSTRACT	8
CONTENIDOS	9
INDICE DE TABLAS	11
CAPÍTULO I: INTRODUCCIÓN Y FUNDAMENTOS DEL PROYECTO	12
1. Formulación del Problema	12
2. Alcances	12
3. Delimitaciones	12
4. Limitaciones	13
5. Objetivos	13
CAPÍTULO II: ESTADO DEL ARTE	14
1. Introducción	14
2. NLP Aplicado a Soporte Técnico	14
3. Bases de Conocimiento como Entrada para Recuperación de Información	15
4. Comparación de Enfoques Vectoriales y Clásicos	15
5. Casos Empresariales Relevantes	16
6. Medidas de evaluación en recuperación de información	16
CAPÍTULO III: METODOLOGÍA	20
1. Recolección y preparación de datos	20
2. Vectorización de contenidos	20
3. Diseño y carga en base de datos vectorial	20
4. Implementación de la lógica de consulta	21

5. Evaluación experimental	21
6. Planificación del proyecto	21
7. Herramientas y tecnologías utilizadas	22
8. Conclusión del Capítulo	23
Capítulo V: Gestión y Selección de la Base de Datos	31
1. Justificación del Uso de una Base de Datos en el Proyecto	31
2. Comparación entre Bases de Datos Relacionales y Vectoriales	31
3. Estado del Arte de la Base de Datos Seleccionada: Weaviate	33
4. Conclusión	35
Referencias	36
Capítulo VI: Selección y Justificación de Modelos de Embedding para Recuperación Semántica	38
Capítulo VII: Creación de embeddings	44
Capítulo VII: Respondiendo preguntas desde la documentación	50
Capítulo VIII: Resultados	52
Capítulo IX: Conclusiones	56
	56
Apéndice A: Modelos de Embedding Relevantes (Hasta principios de 2025)	57
Tabla comparativa de modelos de embedding	57
Referencias Citadas en este Apéndice	62

INDICE DE TABLAS

Tabla 1 - Comparativa de modelos de embedding.....	62
--	----

CAPÍTULO I: INTRODUCCIÓN Y FUNDAMENTOS DEL PROYECTO

1. Formulación del Problema

Las organizaciones enfrentan una creciente demanda en la atención de tickets de soporte técnico, lo que genera una carga operativa significativa en equipos de desarrollo y operaciones. En muchos casos, la respuesta a estos tickets depende exclusivamente del conocimiento tácito de los profesionales, lo que conlleva demoras, respuestas inconsistentes y duplicación de esfuerzos. Esta situación plantea la necesidad de un sistema que facilite la recuperación de información útil desde la documentación técnica existente para asistir a los equipos de soporte de forma automatizada.

2. Alcances

2.1 Alcance Temático

Este proyecto se enmarca en el ámbito del procesamiento de lenguaje natural (NLP) aplicado al soporte técnico, integrando tecnologías como modelos de lenguaje preentrenados y bases de datos vectoriales para la recuperación semántica de información.

2.2 Alcance Temporal

El desarrollo y validación del proyecto se llevará a cabo en un período acotado correspondiente al año académico vigente, con un horizonte de aplicación a corto plazo posterior a su implementación experimental.

3. Delimitaciones

3.1 Delimitación Geográfica

El proyecto simula un entorno de soporte técnico en empresas tecnológicas, pero utiliza exclusivamente datos públicos (documentación de Azure y preguntas en Microsoft Q&A), sin intervención en contextos empresariales privados.

3.2 Delimitación de Tiempo

La validación del sistema considera únicamente un período de pruebas posterior a su implementación, sin incluir la operación continua en un entorno productivo.

4. Limitaciones

El proyecto se ve limitado por la imposibilidad de utilizar tickets reales debido a consideraciones de privacidad y confidencialidad. Asimismo, las fuentes de conocimiento utilizadas pueden incluir artículos que contienen imágenes, diagramas u otro tipo de contenido no procesable mediante técnicas estándar de NLP.

5. Objetivos

5.1 Objetivo General

Desarrollar un sistema de recuperación de información que asista en la resolución de tickets de soporte técnico mediante el uso de procesamiento de lenguaje natural y documentación técnica indexada semánticamente.

5.2 Objetivos Específicos

- Construir un dataset de documentos de soporte técnico a través de un scraper que recolecte datos públicos de preguntas y documentación relevante.
- Implementar una base de datos vectorial que permita procesar y vectorizar los datos utilizando modelos de NLP actuales.
- Implementar la construcción de embeddings con distintos modelos
- Evaluar la precisión del sistema en la recuperación de artículos relevantes frente a tickets simulados.

CAPÍTULO II: ESTADO DEL ARTE

1. Introducción

El avance en el procesamiento de lenguaje natural (NLP, por sus siglas en inglés) ha transformado la forma en que las organizaciones abordan la gestión del conocimiento y el soporte técnico. La integración de tecnologías como modelos de lenguaje, motores de búsqueda vectorial y bases de conocimiento ha permitido desarrollar sistemas más eficientes y escalables. Este capítulo examina el estado actual del arte en relación con la aplicación de NLP al soporte técnico, haciendo énfasis en el uso de bases vectoriales, casos de uso reales y tecnologías emergentes, con el objetivo de contextualizar y fundamentar la solución propuesta en este proyecto.

2. NLP Aplicado a Soporte Técnico

El soporte técnico tradicionalmente ha sido un proceso altamente dependiente del conocimiento humano, lo que conlleva inconsistencias, demoras y errores. La incorporación de técnicas de NLP permite automatizar tareas como la clasificación de tickets, la extracción de intención, la recuperación de respuestas relevantes y la recomendación de soluciones.

Uno de los enfoques más utilizados es el de clasificación automática de tickets, donde modelos como BERT, RoBERTa y DistilBERT han demostrado buenos resultados en tareas de clasificación multiclas y multilabel (Devlin et al., 2018; Liu et al., 2019). Empresas como IBM y SAP han publicado soluciones que incluyen modelos de NLP para análisis semántico de tickets y generación de respuestas automatizadas (Saxena et al., 2021).

Además, el uso de técnicas de resumen automático permite extraer información clave de tickets largos para facilitar la priorización y el enrutamiento inteligente (Gupta & Gupta, 2020).

3. Bases de Conocimiento como Entrada para Recuperación de Información

Una tendencia creciente es la utilización de bases de conocimiento como corpus semántico para alimentar sistemas de recuperación y recomendación en tareas de soporte. Estas bases incluyen artículos técnicos, documentación oficial y respuestas previas.

Los métodos tradicionales de recuperación basados en TF-IDF o BM25 han dado paso a técnicas vectoriales que representan textos como embeddings densos, capturando relaciones semánticas más profundas (Johnson et al., 2021). Esta transición ha sido impulsada por el surgimiento de modelos como Sentence-BERT (Reimers & Gurevych, 2019), que permiten generar representaciones vectoriales eficientes para documentos y consultas.

El uso de retrievers vectoriales se ha vuelto esencial en arquitecturas tipo RAG (Retrieval-Augmented Generation), donde un modelo de lenguaje accede dinámicamente a documentos relevantes durante la generación de una respuesta (Lewis et al., 2020).

4. Comparación de Enfoques Vectoriales y Clásicos

Los sistemas clásicos de recuperación de información, como Lucene y Elasticsearch, utilizan modelos estadísticos que representan documentos como bolsas de palabras. Aunque son eficientes y fáciles de implementar, carecen de comprensión semántica profunda, lo cual limita su capacidad para responder consultas formuladas de forma natural.

En contraste, los sistemas vectoriales utilizan embeddings generados por modelos de aprendizaje profundo, lo cual permite recuperar documentos basados en similitud semántica en lugar de coincidencia léxica (Malkov & Yashunin, 2020). Bases como FAISS, Milvus y Weaviate han surgido como soluciones especializadas en almacenamiento y recuperación de vectores de alta dimensión.

Weaviate, en particular, permite la integración directa con modelos de NLP como OpenAI, Cohere y Hugging Face, facilitando una arquitectura end-to-end desde la ingestión del contenido hasta la consulta semántica (Weaviate, 2023).

5. Casos Empresariales Relevantes

Microsoft

Microsoft ha incorporado modelos de NLP en Azure para análisis automático de tickets y sugerencia de respuestas basadas en documentación técnica. Usan arquitecturas híbridas con embeddings, sistemas de ranking y técnicas de respuesta generativa (Microsoft Learn, 2023).

Zendesk

Zendesk ha desarrollado su propio sistema de inteligencia artificial, "Answer Bot", que utiliza NLP para sugerir artículos de ayuda relevantes al momento en que un usuario envía un ticket. Esto ha reducido en un 10-30% el volumen de tickets que requieren intervención humana (Zendesk, 2023).

ServiceNow

ServiceNow integra modelos de NLP con su módulo "Predictive Intelligence", que clasifica y enruta tickets automáticamente utilizando modelos entrenados en datos históricos. También recomienda artículos de la base de conocimiento (ServiceNow, 2022).

Salesforce

La plataforma Salesforce Service Cloud ha implementado bots que combinan NLP y búsqueda semántica para asistir tanto a clientes como a agentes en tiempo real. Estas herramientas son alimentadas por bases vectoriales generadas a partir de documentación técnica y casos previos (Salesforce, 2023).

6. Medidas de evaluación en recuperación de información

La evaluación de sistemas de recuperación de información (IR) es un aspecto crucial para validar la efectividad del sistema propuesto. En este contexto, las métricas tradicionales de recuperación como precisión, recall, y F1-score siguen siendo ampliamente utilizadas, pero también es necesario considerar métricas específicas que se ajusten al paradigma de recuperación semántica basado en embeddings.

Precisión y Recall

- Precisión (Precision): mide la proporción de documentos relevantes entre los documentos recuperados. Es útil cuando se desea minimizar los falsos positivos, especialmente importante si se busca evitar recomendar artículos irrelevantes al usuario.
- Recall: evalúa la proporción de documentos relevantes recuperados sobre el total de documentos relevantes disponibles. Esta métrica es crucial en el contexto del soporte técnico, donde omitir información relevante puede resultar en una mala atención del ticket.

F1-Score

El F1-score es la media armónica entre precisión y recall. Se utiliza especialmente cuando es importante mantener un equilibrio entre ambas métricas, como es el caso de este proyecto, donde tanto el exceso como la omisión de información relevante pueden afectar la experiencia del usuario final.

Mean Reciprocal Rank (MRR)

MRR es útil cuando se espera que el sistema devuelva una lista ordenada de respuestas y se desea evaluar qué tan pronto aparece la respuesta relevante. En el contexto del soporte técnico, esta métrica es valiosa para evaluar la utilidad de los primeros resultados mostrados al agente.

Normalized Discounted Cumulative Gain (nDCG)

Esta métrica considera no solo la relevancia de los resultados, sino también su posición en la lista. Es útil cuando múltiples respuestas son relevantes, pero se prefiere que las más útiles estén al principio.

Precision@k y *Recall@k*

Ambas métricas están diseñadas para evaluar la calidad de los resultados en sistemas de recuperación que devuelven múltiples documentos ordenados por relevancia.

- *Precision@k* mide la proporción de resultados relevantes entre los primeros k documentos recuperados. Por ejemplo, si entre los primeros 5 artículos sugeridos, 3 son relevantes, entonces $Precision@5 = 3/5 = 0.6$. Esta métrica es útil cuando se busca asegurar que los primeros resultados mostrados sean los más útiles para el usuario.
- *Recall@k*, en cambio, evalúa cuántos documentos relevantes fueron recuperados entre los primeros k , en comparación con el total de documentos relevantes disponibles. Por ejemplo, si hay 4 documentos relevantes en total y el sistema recupera 3 dentro de los primeros 5, entonces $Recall@5 = 3/4 = 0.75$. Esta métrica es fundamental cuando la prioridad es no omitir respuestas potencialmente útiles.

Ambas métricas son complementarias y permiten identificar si un sistema está priorizando documentos relevantes (precisión) y si está capturando una buena proporción de ellos (recall), algo crítico en contextos de soporte técnico (Pinecone, 2023).

Aplicación al Proyecto

En este proyecto se utilizarán principalmente las métricas:

- *Recall@k*: para evaluar cuántas veces se incluye una respuesta correcta dentro de los primeros k resultados.
- *Precision@k*: para validar la calidad de los primeros k documentos sugeridos.

- F1-score: para una evaluación global.
- MRR, cuando se trabaje con rankings semánticos.

Estas métricas permiten evaluar adecuadamente sistemas de recuperación basados en modelos vectoriales y su efectividad en contextos reales de soporte técnico, donde la relevancia y la utilidad inmediata son factores clave.

CAPÍTULO III: METODOLOGÍA

La metodología adoptada en este proyecto se basa en un enfoque aplicado, guiado por los principios del ciclo CRISP-DM y adaptado al contexto de procesamiento de lenguaje natural para soporte técnico. El objetivo es construir un sistema de recuperación semántica que relacione tickets de soporte con documentación técnica relevante. La metodología se compone de las siguientes etapas:

1. Recolección y preparación de datos

- Se utilizarán fuentes públicas como Microsoft Q&A y la documentación técnica de Azure.
- Los datos se extraerán mediante scripts de *scraping* y se almacenarán en archivos estructurados.
- Se realizará limpieza y normalización de texto para garantizar consistencia y eliminar ruido (HTML, metadatos, símbolos).

2. Vectorización de contenidos

- Los textos de preguntas y artículos se vectorizarán utilizando modelos de *embeddings* preentrenados (por ejemplo, *all-MiniLM-L6-v2* de SentenceTransformers).
- Se conservará un identificador único para cada entrada vectorizada.

3. Diseño y carga en base de datos vectorial

- Se empleará Weaviate como motor de búsqueda semántica.

- Se configurará un esquema de clases para representar preguntas, artículos y sus relaciones.
- Se cargarán los vectores y sus metadatos mediante la API de Weaviate.

4. Implementación de la lógica de consulta

- Se diseñarán consultas de búsqueda semántica (*nearText*, *hybrid*, *bm25*) para recuperar documentos relevantes ante un ticket dado.
- Se integrará un mecanismo de evaluación offline para validar los resultados.

5. Evaluación experimental

- Se utilizarán métricas como *Recall@k*, *Precision@k*, *F1-score*, *MRR* y *nDCG*, explicadas en el Capítulo II.
- Se construirán subconjuntos de validación con preguntas etiquetadas manualmente para simular tickets reales.
- Se comparará el rendimiento del sistema en distintos escenarios (con y sin contexto adicional).

6. Planificación del proyecto

La planificación general del desarrollo se distribuye en las siguientes fases:

Fase	Actividad	Duración estimada

1	Definición del problema, objetivos y diseño inicial	2 semanas
2	Recolección de datos desde Azure (Q&A + documentación)	2 semanas
3	Limpieza, preprocesamiento y vectorización	2 semanas
4	Diseño e implementación de base de datos vectorial	2 semanas
5	Integración del sistema y evaluación preliminar	3 semanas
6	Redacción de resultados y validación	3 semanas
7	Ajustes finales y entrega del documento	2 semanas

Cada fase incluye revisión semanal con el profesor guía y ajustes según avance real. Se documentarán resultados intermedios en un cuaderno Jupyter para trazabilidad completa.

7. Herramientas y tecnologías utilizadas

- Lenguaje: Python 3.10
- Librerías: requests, BeautifulSoup, sentence-transformers, pandas, matplotlib
- Motor vectorial: Weaviate (local y cloud)
- Editor de desarrollo: Google Colab / JupyterLab

- Control de versiones: Git + GitHub

8. Conclusión del Capítulo

Este capítulo presentó la metodología que guiará el desarrollo del proyecto, estructurada según el ciclo CRISP-DM. Se abordaron las etapas clave desde la recolección y preparación de datos hasta la evaluación experimental del sistema, con especial énfasis en el uso de técnicas de NLP y almacenamiento vectorial. Asimismo, se definió una planificación detallada que permitirá una implementación organizada y trazable del sistema propuesto. La metodología asegura consistencia entre el planteamiento del problema, los objetivos y la validación experimental que se realizará en etapas posteriores.

El uso de NLP y recuperación semántica en contextos de soporte técnico ha evolucionado significativamente. Empresas como Microsoft, Salesforce y ServiceNow han integrado soluciones híbridas que combinan modelos de lenguaje con bases vectoriales para optimizar sus procesos de atención al cliente (Microsoft Learn, 2023; Salesforce, 2023; ServiceNow, 2022). Esta tendencia refuerza la pertinencia del presente proyecto, que busca aplicar dichos avances en un entorno simulado, utilizando fuentes públicas como preguntas y respuestas de Azure y su documentación técnica.

La elección de tecnologías como Weaviate y modelos de embeddings es coherente con el estado actual de la industria, permitiendo construir un sistema escalable y robusto para la gestión de tickets basado en evidencia científica y buenas prácticas tecnológicas.

CAPÍTULO IV: EXTRACCIÓN AUTOMATIZADA DE DATOS DESDE MICROSOFT LEARN UTILIZANDO SELENIUM Y BEAUTIFULSOUP

1. Introducción

Este capítulo describe el proceso de recopilación automatizada de datos desde Microsoft Learn, como parte del desarrollo del proyecto titulado: **INTEGRACIÓN DE NLP EN LA GESTIÓN DE TICKETS DE SOPORTE TÉCNICO**. En la práctica, los sistemas de soporte técnico cuentan con plataformas donde los usuarios generan tickets de ayuda o consultas, y de forma separada, se dispone de documentación técnica oficial del producto. Este proyecto busca integrar ambas fuentes mediante técnicas de procesamiento de lenguaje natural (NLP), permitiendo, por ejemplo, responder de manera automática preguntas frecuentes a partir de la documentación existente.

Para simular este escenario en un entorno controlado y con acceso público, se utilizó el ecosistema de Azure como caso de estudio. Azure, al ser una plataforma de servicios en la nube ampliamente utilizada, ofrece un repositorio significativo de preguntas técnicas reales a través de su portal Microsoft Q&A, así como una vasta documentación técnica oficial en su sitio web de aprendizaje. Esta dualidad representa fielmente los componentes típicos de un sistema de soporte técnico de productos TI.

2. Herramientas utilizadas

Para la realización de este proceso de extracción de datos, se utilizó la biblioteca **Selenium**, la cual permite automatizar la interacción con navegadores web. Selenium ha sido ampliamente utilizada en tareas de scraping donde el contenido es generado dinámicamente mediante JavaScript (Choudhary, 2019). Complementariamente, se utilizó **BeautifulSoup**, una biblioteca de Python que permite parsear y navegar el árbol DOM para extraer información relevante desde el HTML renderizado.

El script fue implementado en Python y ejecutado de forma local en un entorno con soporte para Google Chrome, utilizando el ejecutable de ChromeDriver.

3. Proceso de scraping

El proceso de scraping se dividió en dos grandes subprocessos:

3.1 Extracción de preguntas y respuestas (Microsoft Q&A)

1. **Captura de enlaces de preguntas:** Se recorrieron todas las páginas indexadas en el tag "Azure" del sitio Microsoft Q&A, obteniendo los enlaces directos a cada una de las preguntas. Esta información fue almacenada en el archivo `question_links.txt`.
2. **Extracción de contenido de cada pregunta:** Utilizando el conjunto de enlaces recolectados, se visitó cada página y se extrajo la siguiente información:
 - Título de la pregunta
 - Fecha de publicación
 - Contenido completo de la pregunta
 - Respuesta aceptada (si existía)
 - Etiquetas o tags asociados
 - Enlace fuente

Cada objeto fue almacenado como una línea en formato JSON dentro del archivo `questions_data.json`.

3.2 Extracción de la documentación oficial de Azure

1. **Captura de enlaces principales:** Se partió desde el índice oficial de Azure (<https://learn.microsoft.com/en-us/azure/>) y se trajeron los enlaces de las secciones principales.
2. **Extracción recursiva de subpáginas:** Para cada página principal se identificaron enlaces internos relevantes, excluyendo páginas que no contenían contenido documental.

3. Extracción del contenido textual: Se obtuvieron los siguientes elementos:

- Título principal del documento
- Resumen introductorio (si está disponible)
- Contenido textual completo de la página (extraído desde el div principal de contenido)
- Enlaces relacionados o hipervínculos a otras secciones

Esta información fue almacenada en un archivo JSON llamado `azure_docs.json`, generando una entrada por documento.

4. Desafíos enfrentados

Durante la ejecución del scraping, se identificaron los siguientes problemas comunes a ambos procesos:

- **Carga asincrónica del contenido:** La página de Microsoft Learn carga ciertos componentes de manera asincrónica, lo que implica que el contenido no está disponible inmediatamente tras la carga inicial. Este problema fue abordado utilizando `WebDriverWait` y condiciones de espera en Selenium.
- **Estructura HTML cambiante:** Se observaron ligeras variaciones en la estructura del HTML entre diferentes preguntas y secciones de documentación. Para resolver esto, se aplicaron selectores CSS robustos y manejo de excepciones.
- **Errores de red y de tiempo de espera:** Algunos enlaces fallaron debido a errores temporales de red o cambios en la disponibilidad del contenido. Estos enlaces fueron almacenados en un archivo `error_links.txt` para su posible reintento posterior.
- **Rendimiento:** Dado el volumen de más de 20.000 preguntas y cientos de documentos, se implementó un sistema incremental que evita reprocesar enlaces ya visitados, guarda avances intermedios y estima el tiempo restante por consola.

5. Estructura del JSON generado

Cada pregunta fue representada como un diccionario con la siguiente estructura:

```
{  
    "title": "How to restrict IP range in Azure NSG policy?",  
    "url": "https://learn.microsoft.com/en-  
us/answers/questions/2242857/...",  
    "question_content": "I want to block any NSG rule that allows  
traffic from 1.2.3.4 or its CIDR range...",  
    "accepted_answer": "You can use this policy definition...",  
    "tags": ["Azure Policy", "NSG", "Security"],  
    "date": "2025-04-01T14:59:36.39+00:00"  
}
```

Y cada documento de Azure fue representado como:

```
{  
    "title": "What is Azure Machine Learning?",  
    "url": "https://learn.microsoft.com/en-us/azure/machine-  
learning/overview",  
    "summary": "Azure Machine Learning is a cloud service for  
accelerating and managing the ML project lifecycle...",  
    "content": "Azure Machine Learning is used for... [contenido  
extenso omitido]",  
    "related_links": ["https://learn.microsoft.com/en-  
us/azure/machine-learning/concept-automated-ml", ...]  
}
```

6. Uso de la IA para la obtención del path del nodo de contenido de las páginas web

En el contexto de la extracción automatizada de información desde páginas web, una tarea clave consiste en identificar con precisión el nodo específico del DOM (Document Object Model) que contiene el contenido principal de interés. Este proceso puede ser complejo debido a variaciones significativas en las estructuras HTML utilizadas en diferentes sitios web o incluso dentro de diferentes páginas de un mismo sitio.

Para resolver eficientemente este problema, se implementó una solución basada en inteligencia artificial utilizando el modelo avanzado Gemini, una tecnología de procesamiento de lenguaje natural desarrollada por Google. Gemini se aplicó con el propósito de analizar diversas páginas del sitio web de documentación técnica de Microsoft Azure y determinar de manera automatizada la mejor estrategia para identificar y obtener el path correcto del nodo que encapsula el contenido relevante.

Este enfoque implicó inicialmente recopilar varias muestras de páginas representativas del sitio web de Azure. Posteriormente, estas muestras fueron analizadas utilizando Gemini para identificar patrones comunes y variaciones significativas en la estructura del contenido principal. A partir de este análisis semántico y estructural, Gemini fue capaz de sugerir los paths XPath óptimos para cada caso, considerando la robustez frente a variaciones estructurales menores y asegurando la consistencia en la extracción.

La ventaja principal de utilizar inteligencia artificial para esta tarea radica en su capacidad para manejar variaciones complejas y adaptarse rápidamente a cambios en el diseño web sin requerir ajustes manuales frecuentes en los scripts de scraping. Esto asegura no solo una mayor precisión en la extracción, sino también una reducción considerable en el tiempo requerido para mantenimiento y actualización de los procesos de extracción automatizada.

En conclusión, la aplicación del modelo Gemini permitió obtener un método efectivo y escalable para identificar de forma automática el path del nodo de contenido adecuado, optimizando así el proceso de extracción de datos desde páginas web dinámicas y variadas, como es el caso de la documentación técnica de Azure.

7. Conclusión

El proceso de extracción descrito en este capítulo constituye una base fundamental para el análisis posterior de datos textuales relacionados con Azure. La combinación de Selenium y BeautifulSoup permitió una extracción efectiva y robusta frente a las complejidades del sitio web objetivo. El conjunto de datos resultante tiene un alto valor para tareas de procesamiento de lenguaje natural, ya que contiene problemas reales, soluciones aceptadas, documentos oficiales y metadatos ricos para cada interacción. Este conjunto de datos permitirá simular escenarios de soporte técnico en los que se pueda vincular la pregunta de un usuario con un fragmento relevante de la documentación oficial.

8. Consideraciones Éticas y Legales del Uso de Documentación Técnica

Durante el desarrollo de este proyecto se utilizó la documentación técnica pública disponible en el portal [learn.microsoft.com](<https://learn.microsoft.com>), específicamente en las secciones relacionadas con productos y servicios de Azure. Para facilitar su análisis y procesamiento automatizado, dicha documentación fue extraída mediante técnicas de web scraping y almacenada en una base de datos vectorial (Weaviate).

Según lo declarado por Microsoft, salvo que se indique lo contrario, los contenidos de Microsoft Learn se encuentran licenciados bajo el esquema [Creative Commons Attribution 4.0 International (CC BY 4.0)](<https://creativecommons.org/licenses/by/4.0/>). Esta licencia permite el uso, adaptación y redistribución del contenido, incluso para fines comerciales, siempre que se otorgue la atribución correspondiente a Microsoft como autor del material original.

En este proyecto, el uso de los contenidos fue estrictamente académico, sin fines de lucro ni redistribución del texto original. Los datos se emplearon como insumo para construir modelos de recuperación semántica de información y evaluar la pertinencia de documentos frente a preguntas formuladas en lenguaje natural. No se publicó ni expuso el contenido textual original en forma íntegra, sino que se utilizaron títulos, fragmentos breves y enlaces, junto con métricas de relevancia para el análisis.

Adicionalmente, se evitó sobrecargar los servidores de origen, se respetaron los límites de acceso establecidos en los headers de cada sitio, y no se emplearon mecanismos que evadan restricciones técnicas (como CAPTCHAs o APIs protegidas).

Se deja constancia de que el uso de esta documentación se encuentra alineado con las condiciones de uso de la plataforma y las buenas prácticas académicas, incluyendo la debida atribución en cada caso correspondiente.

Referencias

Choudhary, A. (2019). *Web Scraping with Python: Collecting Data from the Modern Web*. O'Reilly Media.

Richardson, L. (2007). *Beautiful Soup Documentation*.
<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Selenium Project. (2024). *Selenium with Python*.
<https://selenium.dev/documentation/webdriver/>

CAPÍTULO V: GESTIÓN Y SELECCIÓN DE LA BASE DE DATOS

1. Justificación del Uso de una Base de Datos en el Proyecto

El presente proyecto busca mejorar la eficiencia en la resolución de tickets de soporte técnico reutilizando información técnica preexistente, como documentación oficial y preguntas respondidas en foros especializados. Para lograr esto, se hace necesario almacenar, consultar y analizar grandes volúmenes de texto de manera eficiente y escalable.

En este contexto, el uso de una base de datos es fundamental. En primer lugar, asegura la **persistencia estructurada** de los datos recolectados mediante técnicas de scraping. Además, facilita la gestión de versiones y permite realizar consultas sofisticadas basadas en semántica y metadatos. En palabras de (Stonebraker, 2018), "la base de datos del futuro deberá ser capaz de integrar razonamiento, aprendizaje y recuperación contextual", lo cual es coherente con los objetivos de este proyecto.

Otro aspecto relevante es la **recuperación eficiente de información**. Un sistema inteligente de asistencia a tickets necesita identificar preguntas similares, respuestas existentes y documentación pertinente. Esto requiere una base de datos capaz de operar con representaciones vectoriales del texto, facilitando búsquedas semánticas basadas en similitud contextual.

Finalmente, el almacenamiento en una base de datos facilita la **modularidad del sistema** y la integración con modelos de machine learning, motores de inferencia semántica y visualizaciones interactivas. Esto permite una arquitectura robusta, reutilizable y trazable, esencial en soluciones orientadas a IA aplicada en producción.

2. Comparación entre Bases de Datos Relacionales y Vectoriales

Dado que este proyecto trabaja con representaciones semánticas del texto (embeddings) y necesita realizar búsquedas por similitud, es importante analizar qué tipo de base de datos resulta más adecuada: relacional o vectorial.

2.1 Bases de Datos Relacionales

Las bases relacionales, como PostgreSQL o MySQL, han sido ampliamente utilizadas en sistemas empresariales por su robustez y consistencia. Según Elmasri y Navathe (2016), (Elmasri & Navathe, 2016). Estas bases son ideales para manejar datos estructurados como usuarios, tickets o métricas.

Sin embargo, presentan limitaciones significativas en el manejo de datos semánticos. Si bien es posible almacenar vectores en campos especiales, cualquier operación de similitud debe realizarse fuera del motor de base de datos, lo que reduce el rendimiento y complica el diseño del sistema.

2.2 Bases de Datos Vectoriales

Las bases vectoriales están diseñadas específicamente para hacer búsquedas en espacios de alta dimensión. FAISS, Milvus y Weaviate son algunos ejemplos. Según Johnson et al. (2019), "el uso de FAISS permite realizar búsquedas por similitud entre millones de vectores de forma eficiente usando GPU", lo que las hace ideales para sistemas de recuperación de información semántica.

Estas bases permiten integrar directamente modelos de lenguaje que generan vectores de texto, almacenarlos y luego realizar consultas del tipo "encuentra los 10 textos más similares a este embedding". Además, algunas permiten añadir filtros por metadatos, uniendo semántica y lógica relacional.

2.3 Comparación y Elección

Criterio	Relacional (PostgreSQL, MySQL)	Vectorial (FAISS, Milvus, Weaviate)
----------	-----------------------------------	--

Eficiencia en consultas semánticas	Baja	Alta
Soporte a metadatos	Alta	Media/Alta
Escalabilidad en vectores	Baja	Alta
Integración con NLP	Requiere lógica externa	Nativa o vía módulos
Madurez tecnológica	Alta	Media
Comunidad y soporte	Amplia	En crecimiento

La elección ideal para este proyecto es una base vectorial. Entre las disponibles, **Weaviate** destaca por su arquitectura híbrida (vector + metadatos), integración con modelos de lenguaje como OpenAI y HuggingFace, y soporte para consultas mediante GraphQL. SeMI Technologies (2023) destaca que "Weaviate permite realizar búsquedas híbridas, combinando semántica y filtros booleanos, con una API altamente expresiva".

Además, según Agarwal, Sethi y Varma (2022), "la combinación de vectores semánticos con filtros estructurados permite recuperar información de manera más precisa y contextual que los enfoques tradicionales".

3. Estado del Arte de la Base de Datos Seleccionada: Weaviate

Weaviate es una base de datos vectorial moderna, open-source, diseñada para almacenar objetos enriquecidos con vectores semánticos y metadatos.

3.1 Arquitectura y Funcionamiento

Weaviate utiliza un índice HNSW (Hierarchical Navigable Small World) que, como afirman Malkov y Yashunin (2018), "permite realizar búsquedas por vecinos más cercanos de forma rápida y precisa, incluso con millones de vectores". Cada objeto en Weaviate se compone de:

- Un vector de embedding generado automáticamente o cargado por el usuario.
- Un conjunto de metadatos estructurados (como título, URL, fecha).
- Posibles relaciones con otros objetos.

Esta estructura permite consultas del tipo: "encuentra los cinco artículos más similares a este ticket, publicados en los últimos seis meses".

Weaviate puede operar en entornos locales o en la nube, integrándose con motores de embeddings a través de módulos nativos. Esto agiliza el flujo de datos desde el texto original hasta la consulta por similitud.

3.2 Casos de Uso y Adopción

Weaviate se ha utilizado en contextos similares al de este proyecto. Por ejemplo, Deutsche Telekom lo aplica en su sistema de soporte técnico automatizado. Según Weaviate Blog (2023), "la base de datos vectorial permite recuperar automáticamente respuestas similares a tickets anteriores, reduciendo el tiempo de atención".

Además, su adopción en proyectos de búsqueda semántica, asistentes virtuales y clasificación inteligente de texto lo consolidan como una herramienta de referencia en la industria de PLN.

3.3 Ventajas Técnicas Relevantes

- Consultas por GraphQL, permitiendo combinar semántica y lógica condicional.
- Soporte para múltiples backends y ejecución distribuida.
- Integración con modelos de lenguaje populares.
- Licencia open-source, lo que facilita la adaptación y despliegue en distintos entornos.

3.4 Limitaciones

Algunas de sus limitaciones son:

- Curva de aprendizaje inicial para diseñar esquemas de clases y relaciones.
- Costos computacionales al realizar inserciones masivas si no se optimizan procesos.
- Requiere una capa de backend adicional para tareas de administración, monitoreo o visualización.

A pesar de esto, Weaviate representa la solución más alineada con las necesidades del proyecto, dada su capacidad para manejar búsquedas semánticas complejas con alta precisión y rendimiento.

4. Conclusión

La elección de una base de datos no es un aspecto técnico aislado, sino una decisión estratégica. En el caso de este proyecto, que requiere búsquedas

semánticas eficientes sobre grandes volúmenes de texto técnico, Weaviate representa la solución más adecuada.

Su arquitectura híbrida, su compatibilidad con modelos de lenguaje modernos y su capacidad de realizar búsquedas combinadas por similitud y filtros estructurados hacen de Weaviate un motor ideal para apoyar sistemas de asistencia inteligente en soporte técnico.

Complementar Weaviate con una base relacional ligera para métricas y seguimiento administrativo permite construir un sistema robusto, escalable y modular, alineado con los objetivos de eficiencia y reutilización del conocimiento planteados desde el inicio.

Referencias

- Agarwal, A., Sethi, A., & Varma, M. (2022). "Hybrid information retrieval systems using vector and relational databases". *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*.
- Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.
- Johnson, J., Douze, M., & Jégou, H. (2019). "Billion-scale similarity search with GPUs". *IEEE Transactions on Big Data*, 7(3), 535-547.
- Malkov, Y. A., & Yashunin, D. A. (2018). "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824–836.
- SeMI Technologies. (2023). *Weaviate Documentation*. <https://weaviate.io>

- Stonebraker, M. (2018). "What Does the Database of the Future Look Like?" *Communications of the ACM*, 61(2), 36–39.

- Weaviate Blog. (2023). *Case Studies*. <https://weaviate.io/blog>

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7^a ed.). Pearson.

Stonebraker, M. (2018). What Does the Database of the Future Look Like?
Communications of the ACM, 61(2), 36–39.

CAPÍTULO VI: SELECCIÓN Y JUSTIFICACIÓN DE MODELOS DE EMBEDDING PARA RECUPERACIÓN SEMÁNTICA

1. Introducción a la Estrategia de Vectorización

La piedra angular de un sistema eficaz de recuperación de información semántica, como el que se propone en este proyecto, reside en la calidad y adecuación de las representaciones vectoriales —o *embeddings*— del texto. Un embedding robusto debe encapsular con precisión el significado, el contexto y los matices semánticos de los documentos técnicos y las consultas de los usuarios. Esto permite que el sistema identifique y recupere los artículos más pertinentes en respuesta a una pregunta formulada en lenguaje natural, incluso cuando no exista una coincidencia exacta de palabras clave. La elección del modelo para generar estos vectores es una decisión crítica, ya que influye directamente en la precisión de la recuperación, la eficiencia computacional del sistema y su capacidad para escalar y adaptarse a las necesidades del soporte técnico.

Este capítulo se dedica a fundamentar la selección de modelos específicos para la tarea de vectorización de la documentación técnica y las preguntas de soporte. Se explorarán las familias de modelos **Sentence-BERT (S-BERT)** y **E5 (Embeddings from Language Models)** como alternativas de código abierto controladas por el proyecto. Adicionalmente, se establece una comparación entre esta estrategia de vectorización personalizada y el uso del motor de vectorización por defecto que ofrece Weaviate, específicamente `text2vec-openai`, que utiliza los modelos de OpenAI. El objetivo es destacar las ventajas inherentes a un enfoque adaptado y controlado, y al mismo tiempo, considerar la potencia de los modelos propietarios de última generación.

2. Justificación de los Modelos de Embedding Seleccionados

Para el desarrollo de este proyecto, se ha determinado la evaluación comparativa de dos familias principales de modelos de embedding de código abierto, además de los modelos propietarios accesibles a través de Weaviate: **Sentence-BERT (S-BERT)** y **E5**.

2.1. Sentence-BERT (S-BERT)

La utilización de modelos pertenecientes a la arquitectura S-BERT, con un enfoque particular en variantes como all-MiniLM-L6-v2, resulta apropiada por varias razones. La preferencia por S-BERT sobre otros modelos de lenguaje de transformadores, como BERT base o DistilBERT, para la generación directa de embeddings de similitud, se sustenta en su optimización explícita para estas tareas (Reimers & Gurevych, 2019).

- **Optimización Específica para Similitud Semántica:** A diferencia de los modelos BERT tradicionales, que fueron concebidos y entrenados principalmente para tareas de clasificación de texto o análisis a nivel de token, los modelos S-BERT han sido afinados específicamente para generar embeddings de frases y documentos. Estas representaciones vectoriales están diseñadas para ser comparadas directamente mediante métricas de similitud, como la similitud coseno, para evaluar la cercanía semántica (Reimers & Gurevych, 2019). Esta característica es crucial para el objetivo central del proyecto. Investigaciones más recientes continúan validando la efectividad de los embeddings a nivel de oración para tareas de recuperación semántica (Muennighoff et al., 2023).
- **Eficiencia Computacional y Rendimiento:** Los modelos S-BERT, y en particular variantes optimizadas como all-MiniLM-L6-v2 —derivada de arquitecturas eficientes como MiniLM (Wang et al., 2020)—, están diseñados para ofrecer un equilibrio notable entre la calidad de la representación semántica y la eficiencia computacional. Son significativamente más ligeros y rápidos que sus contrapartes BERT más grandes, sin incurrir en una pérdida drástica de rendimiento en tareas de similitud. La comunidad de Hugging Face (s.f.-a) mantiene y evalúa continuamente estos modelos, proporcionando benchmarks actualizados sobre su rendimiento.
- **Autonomía, Control y Transparencia:** La utilización de un modelo de embedding de código abierto S-BERT, que puede ser alojado y ejecutado

localmente, otorga un grado completo de control sobre todo el proceso de vectorización.

2.2. E5 (Embeddings from Language Models)

Como una alternativa más reciente y con un rendimiento destacado en benchmarks, se ha seleccionado la familia de modelos E5 (Wang et al., 2022). Estos modelos, desarrollados por Microsoft Research, ofrecen varias ventajas:

- **Rendimiento Sobresaliente en Benchmarks:** Los modelos E5 han demostrado consistentemente un alto rendimiento en el benchmark MTEB (Massive Text Embedding Benchmark) (Muennighoff et al., 2023), superando a muchos modelos anteriores en diversas tareas de comprensión y recuperación de texto. Esto sugiere una capacidad robusta para capturar matices semánticos.
- **Preentrenamiento Contrastivo Efectivo:** E5 utiliza una estrategia de preentrenamiento contrastivo sobre pares de textos (consulta, pasaje) obtenidos de diversas fuentes, lo que les permite aprender representaciones densas muy efectivas para la búsqueda semántica. Este enfoque ha demostrado ser particularmente útil para tareas de recuperación de información donde la relevancia no siempre se basa en la coincidencia léxica directa.
- **Disponibilidad en Diversos Tamaños y Código Abierto:** Al igual que S-BERT, los modelos E5 están disponibles en diferentes tamaños (e.g., e5-small, e5-base, e5-large, y versiones v2) a través de la plataforma Hugging Face (Hugging Face, s.f.-c). Esto permite un balance entre la calidad del embedding y los recursos computacionales necesarios, y su naturaleza de código abierto se alinea con los objetivos de transparencia y control del proyecto.

La inclusión de S-BERT y E5 permite una evaluación exhaustiva de modelos de embedding de código abierto con diferentes arquitecturas y estrategias de entrenamiento, proporcionando una base sólida para la comparación con los modelos de OpenAI.

3. Comparación Estratégica con la Vectorización Nativa de Weaviate (text2vec-openai)

Weaviate, como base de datos vectorial, ofrece una integración conveniente y potente con diversos módulos de vectorización, entre los que destaca text2vec-openai. Este módulo permite la creación automática de embeddings utilizando los modelos de lenguaje avanzados de OpenAI (OpenAI, 2024a), lo cual representa una solución robusta y de fácil implementación. Sin embargo, para los objetivos específicos y las necesidades de optimización de este proyecto, se consideró que un enfoque de vectorización personalizado y controlado externamente (utilizando S-BERT o E5) ofrecía beneficios sustanciales a evaluar.

- **Adaptación al Contexto Específico vs. Generalización:** Los modelos proporcionados por OpenAI (2024a, 2024b), si bien son extremadamente potentes y versátiles (e.g., la familia de modelos text-embedding-3), están entrenados sobre corpus masivos y diversos, lo que los hace modelos de propósito general. En el contexto de la documentación técnica de Azure, que posee una jerga, estructura y tipos de contenido muy específicos, un modelo S-BERT o E5 puede ser afinado o seleccionado por su rendimiento en dominios similares. Más importante aún, al controlar externamente la generación de embeddings, podemos experimentar con diferentes estrategias de combinación de los campos textuales disponibles (title, summary, content). Esta flexibilidad permite crear un embedding que pondere diferencialmente la importancia de cada sección del documento. Por ejemplo, el título y el resumen suelen contener la información más concisa y relevante para una consulta inicial. Este nivel de granularidad y ajuste fino sobre el texto que se vectoriza no es directamente controlable con la misma facilidad cuando se utiliza el módulo text2vec-openai, que típicamente procesa el contenido textual proporcionado como un todo unificado.
- **Potencial de Optimización del Rendimiento y Precisión en la Recuperación:** Una hipótesis central de este proyecto es que un embedding construido a partir de una combinación estratégica y ponderada de los campos de texto más significativos de un documento

técnico puede superar en precisión a un embedding genérico para la tarea específica de recuperación de documentación. Al generar y evaluar vectores para configuraciones como solo title, title + summary, y title + summary + content utilizando S-BERT y E5, podemos determinar empíricamente qué representación textual captura de manera más efectiva la intención subyacente de la consulta del usuario y la esencia semántica del documento técnico. Esto permite una optimización directa de las métricas de evaluación clave para sistemas de recuperación, tales como **Recall@k**, **Precision@k** y **Mean Reciprocal Rank (MRR)**, tal como se explora en evaluaciones comparativas de embeddings (e.g., Liu et al., 2023).

- **Transparencia, Replicabilidad y Costo:** El uso de modelos de código abierto como all-MiniLM-L6-v2 (Wang et al., 2020; Hugging Face, s.f.-b) y los modelos E5 (Wang et al., 2022; Hugging Face, s.f.-c) asegura una completa transparencia en el proceso de generación de embeddings. Los pesos del modelo, su arquitectura y su código de inferencia son accesibles, lo que facilita la comprensión de su funcionamiento y la replicabilidad de los resultados experimentales. Esto es fundamental en el contexto de un proyecto de investigación académica, donde la validación independiente y la trazabilidad de la metodología son primordiales. Adicionalmente, el uso de modelos locales elimina los costos directos asociados al consumo de APIs de modelos propietarios, lo que puede ser una consideración importante para la sostenibilidad y escalabilidad del sistema a largo plazo.

4. Conclusión del Capítulo

La decisión de emplear y comparar modelos de las familias **Sentence-BERT** (Reimers & Gurevych, 2019) y **E5** (Wang et al., 2022) para la generación de los embeddings textuales, y la estrategia de experimentar con diversas combinaciones de los campos title, summary y content, no es una elección arbitraria, sino una decisión metodológica fundamentada. Este enfoque permite la construcción y evaluación de representaciones vectoriales que están altamente

especializadas y potencialmente optimizadas para el dominio específico del soporte técnico de la documentación de Azure.

Si bien la funcionalidad de vectorización automática que provee Weaviate a través de módulos como text2vec-openai (utilizando los modelos de OpenAI) es indudablemente poderosa y simplifica la implementación, el control granular, la capacidad de ajuste fino, la transparencia y el potencial de optimización del rendimiento que ofrece nuestro enfoque personalizado con modelos de código abierto son factores determinantes a investigar. Estos elementos son cruciales para maximizar la relevancia y precisión de los resultados del sistema de recuperación de información y, en última instancia, para cumplir con el objetivo general del proyecto: desarrollar un sistema de asistencia eficaz y preciso para la gestión de tickets de soporte técnico, basado en la recuperación semántica de conocimiento existente. La comparación empírica entre estos tres enfoques (S-BERT, E5 y OpenAI) proporcionará información valiosa sobre la mejor estrategia de vectorización para el problema planteado.

Referencias

- Hugging Face. (s.f.-a). *Sentence Transformers Library*. Recuperado el 24 de mayo de 2025, de <https://www.sbert.net>
- Hugging Face. (s.f.-b). *all-MiniLM-L6-v2 Model Card*. Recuperado el 24 de mayo de 2025, de <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>
- Hugging Face. (s.f.-c). *Microsoft E5 Text Embeddings*. Recuperado el 24 de mayo de 2025, de https://huggingface.co/microsoft?search_models=e5 (Nota: Esta es una página general de búsqueda; se recomienda citar el model card específico de la variante E5 utilizada, e.g., intfloat/e5-large-v2).
- Liu, Y., Yao, Y., Ton, J.-F., Wang, X., Wang, S., Cheng, W., & Xiang, B. (2023). *MTEB: Massive Text Embedding Benchmark*. arXiv preprint arXiv:2210.07316. <https://arxiv.org/abs/2210.07316>

- Muennighoff, N., Tazi, N., Magne, L., & Reimers, N. (2023). *MTEB: Massive Text Embedding Benchmark*. Journal of Machine Learning Research, 24(264), 1-55. <https://jmlr.org/papers/v24/22-1267.html>
- OpenAI. (2024a). *Embeddings*. OpenAI API Documentation. Recuperado el 24 de mayo de 2025, de <https://platform.openai.com/docs/guides/embeddings>
- OpenAI. (2024b, 25 de enero). *New embedding models and API updates*. OpenAI Blog. Recuperado el 24 de mayo de 2025, de <https://openai.com/blog/new-embedding-models-and-api-updates>
- Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. En *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 3982–3992). Association for Computational Linguistics. <https://doi.org/10.18653/v1/D19-1410>
- Wang, L., Yang, N., Huang, J., Chang, M. W., & Wang, W. (2022). *Text Embeddings by Weakly-Supervised Contrastive Pre-training*. arXiv preprint arXiv:2212.03533. <https://arxiv.org/abs/2212.03533> (Este es el paper principal de los modelos E5).
- Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). *MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers*. arXiv preprint arXiv:2002.10957. <https://arxiv.org/abs/2002.10957>

CAPÍTULO VII: CREACIÓN DE EMBEDDINGS

1. Introducción al Proceso de Vectorización

Una vez seleccionados y justificados los modelos de embedding en el capítulo anterior, el siguiente paso crucial es la implementación del proceso de vectorización. Este capítulo detalla la metodología técnica seguida para transformar el corpus de texto —compuesto por la documentación de Azure y las preguntas de Microsoft Q&A— en representaciones vectoriales numéricas. Estos

vectores, o embeddings, son el pilar del sistema de recuperación semántica, ya que permiten realizar búsquedas por similitud contextual en lugar de depender exclusivamente de coincidencias léxicas (Reimers & Gurevych, 2019; Muennighoff et al., 2023).

El proceso fue diseñado como una fase experimental controlada, comparando modelos de código abierto como S-BERT (all-MiniLM-L6-v2) y E5 (e5-base, e5-large-v2) con el modelo propietario de OpenAI (text-embedding-3-large), evaluando además distintas estrategias de combinación textual: título, resumen, contenido, o combinaciones de estos. Esta experimentación tiene como objetivo identificar cuál configuración produce los vectores más representativos para el dominio técnico de Azure y sus consultas de soporte (Wang et al., 2022).

2. Preparación del Entorno y de los Datos

El entorno técnico se implementó sobre Python 3.10, utilizando plataformas como Google Colab y JupyterLab, lo que permitió acceso a GPU y un entorno flexible para pruebas. Se emplearon las librerías pandas para la manipulación de datos, sentence-transformers para la generación local de vectores con modelos de Hugging Face (Hugging Face, s.f.-a), y el cliente oficial de Weaviate para la ingesta posterior de datos en la base vectorial (SeMI Technologies, 2023).

Los datos provenientes del proceso de scraping ya habían sido cargados previamente en dos colecciones de Weaviate denominadas Documentation (para los artículos técnicos de Azure) y QnA (para las preguntas de Microsoft Q&A). Por tanto, el flujo de trabajo comenzó consultando directamente estas colecciones desde Python para extraer y preparar los textos necesarios para la vectorización. Esta arquitectura permitió una integración más fluida entre los modelos de embedding y la base vectorial.

3. Estrategias de Composición de Texto para Vectorización

Una hipótesis clave de esta investigación es que no todos los fragmentos de un documento aportan por igual al significado global del texto desde la perspectiva

de recuperación semántica. Por ello, se definieron y aplicaron cuatro estrategias de composición:

- **E1: Solo title** – captura una representación muy concisa del documento.
- **E2: title + summary** – representa el núcleo del contenido con contexto adicional.
- **E3: title + summary + content** – ofrece una visión exhaustiva.
- **E4: Solo content** – permite analizar el valor del cuerpo textual por sí solo.

Para las preguntas del dataset QnA, se utilizó la combinación `title + question_content`, simulando la formulación típica de una consulta en lenguaje natural.

Estas estrategias fueron aplicadas a cada modelo de embedding, permitiendo una matriz cruzada de experimentación que posteriormente será evaluada en función de métricas de recuperación como `Recall@k` y `MRR`.

4. Generación de Embeddings con los Modelos Seleccionados

4.1 Modelos de Código Abierto (S-BERT y E5):

Los modelos fueron cargados desde Hugging Face (`sentence-transformers/all-MiniLM-L6-v2`, `intfloat/e5-base`, `intfloat/e5-large-v2`) y ejecutados localmente utilizando `sentence-transformers`, lo que permitió un control detallado sobre la tokenización, segmentación de textos largos (en caso de superar los 512 tokens) y uso de `mean pooling` para generar los vectores finales (Reimers & Gurevych, 2019).

TODO: Insertar código que muestre la tokenización, segmentación y pooling con `sentence transformers`

Cada vector fue almacenado junto con los siguientes metadatos: ID del documento, modelo utilizado, estrategia textual aplicada y texto vectorizado. Esto permitió no solo la trazabilidad, sino también la implementación de múltiples variantes por documento.

4.2 Modelo Propietario (OpenAI vía Weaviate):

En paralelo, se habilitó la vectorización automática a través del módulo `text2vec-openai` de Weaviate, el cual invoca la API de OpenAI para generar embeddings del contenido completo de cada documento. Esta opción simplifica la implementación, aunque limita la granularidad del control sobre la composición textual (OpenAI, 2024b).

TODO: Agregar código para el esquema de definición de clases y configuración de vectorizador automático de weaviate

5. Estructuración y Carga en la Base de Datos Vectorial Weaviate

Se diseñó un esquema basado en una clase `DocumentationEmbedding`, independiente de la clase original `Documentation`, para almacenar embeddings generados manualmente con modelos externos. Esta clase incluyó los siguientes campos:

- `doc_id`: identificador del documento original
- `model`: nombre del modelo (`sbert`, `e5-base`, `e5-large`, `openai`)
- `strategy`: `E1`, `E2`, `E3` o `E4`
- `text`: fragmento textual usado para la vectorización
- `vector`: embedding generado
- `timestamp`: marca de tiempo de inserción

TODO: Insertar código para la creación de objetos en lotes con vectores personalizados

Para los modelos generados localmente, se utilizó el método `batch.data.objects.create()` del cliente de Python de Weaviate, deshabilitando la vectorización automática (`vectorizer=None`). En el caso del modelo de OpenAI, los objetos fueron insertados directamente y vectorizados automáticamente por el módulo habilitado.

Esta organización de la base de datos permitió comparar entre múltiples representaciones vectoriales para un mismo documento, sentando las bases para una experimentación sistemática y replicable.

6. Conclusión del Capítulo

Este capítulo ha descrito el proceso práctico y técnico de transformar datos textuales en embeddings consultables, utilizando diversas estrategias de composición textual y modelos de última generación, tanto de código abierto como propietarios. La metodología aplicada garantiza la reproducibilidad, comparabilidad y extensibilidad del sistema, facilitando su evaluación futura.

Se construyó una base vectorial estructurada, con múltiples versiones embebidas de cada documento. Este enfoque no solo permite elegir el mejor modelo-estrategia, sino también habilita futuras líneas de investigación, como el entrenamiento de rerankers o el análisis de bias semántico por tipo de texto.

Referencias

Hugging Face. (s.f.-a). Sentence Transformers Library. Recuperado el 25 de mayo de 2025, de <https://www.sbert.net>

Hugging Face. (s.f.-b). all-MiniLM-L6-v2 Model Card. Recuperado el 25 de mayo de 2025, de <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

Muennighoff, N., Tazi, N., Magne, L., & Reimers, N. (2023). MTEB: Massive Text Embedding Benchmark. Journal of Machine Learning Research, 24(264), 1-55. <https://jmlr.org/papers/v24/22-1267.html>

OpenAI. (2024a). Embeddings. OpenAI API Documentation. Recuperado el 25 de mayo de 2025, de <https://platform.openai.com/docs/guides/embeddings>

OpenAI. (2024b, 25 de enero). New embedding models and API updates. OpenAI Blog. Recuperado el 25 de mayo de 2025, de <https://openai.com/blog/new-embedding-models-and-api-updates>

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. In Proceedings of EMNLP-IJCNLP 2019 (pp. 3982–3992). <https://doi.org/10.18653/v1/D19-1410>

SeMI Technologies. (2023). Weaviate Documentation. Recuperado el 25 de mayo de 2025, de <https://weaviate.io>

Wang, L., Yang, N., Huang, J., Chang, M. W., & Wang, W. (2022). Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv preprint arXiv:2212.03533. <https://arxiv.org/abs/2212.03533>

CAPÍTULO VII: RESPONDiendo PREGUNTAS DESDE LA DOCUMENTACIÓN

Usando Op0enAI combining title + content

 Preguntas evaluadas: 3089

Precision@5: 0.034

Recall@5: 0.112

MRR@5: 0.072

 Tiempo total de evaluación: 1197.07 segundos

 Tiempo promedio por pregunta: 0.39 segundos

Using only content:

 Preguntas evaluadas: 3089

Precision@5: 0.030

Recall@5: 0.103

MRR@5: 0.067

 Tiempo total de evaluación: 1173.61 segundos

 Tiempo promedio por pregunta: 0.38 segundos

CAPÍTULO VIII: RESULTADOS

La evaluación del sistema de recuperación semántica propuesto se llevó a cabo utilizando un conjunto de 3.089 preguntas reales provenientes de foros técnicos, cada una con su respectiva respuesta aceptada que incluye, en muchos casos, vínculos directos a la documentación oficial de Microsoft. Esta información permitió establecer un criterio objetivo de validación, comparando si los documentos sugeridos por el sistema incluían aquellos enlaces aceptados por los usuarios como respuestas correctas.

8.1 Metodología de evaluación

Se utilizó una estrategia de búsqueda basada en vectores para representar tanto las preguntas como los documentos técnicos. En el caso de las preguntas, se vectorizó siempre el campo `question_content`. Por su parte, los documentos fueron representados a través de diferentes combinaciones de sus campos: `title`, `summary` y `content`. Estas combinaciones fueron previamente embebidas mediante el modelo `all-MiniLM-L6-v2` y almacenadas en una colección separada en Weaviate.

Se evaluaron cinco variantes de representación documental: `title`, `title+summary`, `content`, `title+content`, y `title+summary+content`. Además, se compararon los resultados obtenidos con el modelo MiniLM con los generados por el vectorizador `text2vec-openai` integrado en Weaviate, utilizando como referencia únicamente el contenido (`content`) o la combinación `title+content`.

Las métricas empleadas para la evaluación fueron `Precision@5`, `Recall@5`, `MRR@5`, `F1-score` y `nDCG`, enfocándose en el top-5 documentos retornados por cada consulta. También se midió el tiempo total y promedio requerido por pregunta.

8.2 Resultados utilizando MiniLM

Los resultados obtenidos al utilizar el modelo MiniLM para vectorizar las preguntas y compararlas contra distintas representaciones documentales se resumen en la siguiente tabla:

input_type	nDCG	Recall	Precisión	MRR	F1	Tiempo total (s)	Tiempo prom. (s)
title	0.0502	0.0893	0.0272	0.0447	0.0393	334.04	0.1156
title+summary	0.0483	0.0897	0.0278	0.0441	0.0399	319.52	0.1116
content	0.0578	0.0685	0.0216	0.0512	0.0308	318.44	0.1080
title+content	0.0630	0.0779	0.0241	0.0550	0.0347	321.87	0.1087
title+summary+content	0.0649	0.0833	0.0256	0.0573	0.0370	322.12	0.1084

Se observa que la mejor representación documental corresponde a la combinación title+summary+content, superando al resto de las variantes en todas las métricas principales. Aunque su ventaja en recall y precisión es marginal respecto a title+summary, muestra una mejora más clara en MRR y nDCG, lo que sugiere que los documentos relevantes aparecen más arriba en el ranking.

Por otro lado, utilizar únicamente el contenido (content) disminuye significativamente tanto la precisión como el recall, lo que evidencia que los campos title y summary aportan información semántica valiosa para la tarea de recuperación.

8.3 Resultados utilizando text2vec-openai

Con el objetivo de comparar el desempeño con el vectorizador por defecto de Weaviate, se evaluaron dos configuraciones adicionales utilizando text2vec-openai:

- Usando solo el campo content.
- Usando la combinación title + content.

Los resultados fueron los siguientes:

input_ty	Preciso pe	n@5	Recall @5	MRR @5	Tiempo total (s)	Tiempo prom. (s)
content	0.030	0.103	0.067	1173.61	0.38	
title+content	0.034	0.112	0.072	1197.07	0.39	

Comparado con MiniLM, text2vec-openai muestra métricas ligeramente superiores, especialmente en Recall y MRR. Esto indica una mayor capacidad de recuperación y mejor posicionamiento de los documentos relevantes en los primeros resultados. Sin embargo, también presenta un costo computacional más alto, tanto en tiempo total como promedio.

8.4 Comparación general y conclusiones

Ambas estrategias logran un desempeño razonable considerando el entorno complejo de evaluación, en donde solo se considera como documento relevante aquel cuya URL está incluida explícitamente en una respuesta aceptada por la comunidad. Esta metodología introduce una exigencia adicional respecto a benchmarks estándar como BEIR, pero resulta más realista en un contexto de soporte técnico automatizado.

El modelo MiniLM ofrece un buen balance entre eficiencia y rendimiento, especialmente cuando se utilizan representaciones documentales enriquecidas

como title+summary+content. Por otro lado, text2vec-openai presenta un rendimiento ligeramente mejor en términos de recuperación, lo que lo hace una alternativa válida si se dispone de mayor capacidad computacional o se prioriza la calidad de resultados.

CAPÍTULO IX: CONCLUSIONES

APÉNDICE A: MODELOS DE EMBEDDING RELEVANTES (HASTA PRINCIPIOS DE 2025)

Este apéndice proporciona una visión general de varios modelos y familias de modelos de embedding de texto que son relevantes en el panorama actual del Procesamiento del Lenguaje Natural (NLP) y la recuperación de información. La elección de un modelo de embedding específico a menudo depende de un equilibrio entre el rendimiento en la tarea, la eficiencia computacional, el costo y la facilidad de implementación.

El campo evoluciona rápidamente, y los benchmarks públicos como MTEB (Massive Text Embedding Benchmark), alojado en Hugging Face, son un recurso crucial para seguir y comparar el rendimiento de los modelos en una amplia gama de tareas.

Tabla comparativa de modelos de embedding

Modelo	Tipo	Puntuación	Descripción breve	Pro	Contra
MTEB					
NV-Embed-v2	Propietario	69.32	Modelo de Rendimiento NVIDIA basado o superior en Mistral 7B, líder en MTEB.	Requiere recursos computacionales significativos.	múltiples tareas.
Voyage-Large-2-Instruct	Propietario	68.28	Modelo de Excelente instrucción para recuperación de información.	Menos accesible para uso general.	tareas de recuperación de información.

recuperaci
ón.

Linq- Embed- Mistral	Propietario	68.17	Modelo basado en Mistral optimizado para embeddings.	Alto rendimiento en o benchmark a específica.	Dependencia de infraestructura.
----------------------------	-------------	-------	--	---	---------------------------------

SFR- Embeddi- ng-Mistral	Propietario	67.56	Modelo de Salesforce para recuperación de información.	Buen desempeño en tareas de recuperación.	Menos versátil en otras tareas.
--------------------------------	-------------	-------	--	---	---------------------------------

GTE- Qwen- 1.5-7B- Instruct	Open Source	67.34	Modelo de Alibaba con capacidades multilingües y multitarea.	Soporte múltiples idiomas y para tareas.	Requiere recursos significativos y para entrenamiento.
--------------------------------------	----------------	-------	--	--	--

BGE-M3- Embeddi- ng	Open Source	67.00	Modelo de BAAI con enfoque en multilingüismo y	Versátil líder en benchmark	y Mayor complejidad en s.
---------------------------	----------------	-------	--	-----------------------------	---------------------------

				multifuncionalidad.	implementación.
E5-Mistral-7B-Instruct	Open Source	66.50	Modelo Microsoft basado en Mistral para recuperación.	Excelente en recuperació n y clasificación.	Alto costo computacional.
Nomic-Embed-Text-v1.5	Open Source	65.00	Modelo Nomic AI con enfoque en transparencia y reproducibilidad.	Totalmente abierto y reproducible.	Rendimiento inferior frente a líderes del benchmark.
Instructor-XL	Open Source	64.50	Modelo que genera embeddings condicionados por instrucciones.	Adaptabilidad a tareas específicas bien formuladas.	Requiere instrucciones bien formuladas.
OpenAI text-rio	Propietario	64.00	Modelo de OpenAI accesible vía API con alto	Fácil integración y uso.	Dependencia del proveedor.

embedding-3-large

rendimiento general.

y costos asociados.

Cohere-embed-english-v3

Propietario

63.50

Modelo Cohere optimizado para empresas.

de Alto rendimiento personalizació.

Menor control sobre empresarial.

Mistral-Embed

Propietario

63.00

Modelo Mistral optimizado para recuperación aumentada.

de Buen AI rendimiento para tareas específicas.

Menos opciones para uso local.

BERT

Open Source

60.00

Modelo preentrenado bidireccional basado en transformers.

de Muy versátil para múltiples tareas de NLP.

No optimizado para generación directa de embeddings.

DistilBERT

Open Source

59.00

Versión reducida de BERT.

Rápido y eficiente.

Menor precisión en tareas.

				semánticas profundas.
RoBERTa	Open Source	61.00	Variante robusta de BERT entrenada con más datos.	Mejora sobre BERT. Más costoso en recursos.
MPNet	Open Source	62.00	Modelo Microsoft para dependencias semánticas.	Buen rendimiento o semántico. Menos adoptado.
ST5	Open Source	63.00	Versión de T5 entrenada como generador de embeddings.	Flexible y multitarea. Alto consumo de recursos.
MiniLM	Open Source	58.00	Modelo optimizado para eficiencia.	Excelente dispositivo para dispositivos con capacidades semánticas más limitadas. Capacidad menor que los modelos más grandes.

SBERT	Open Source	60.50	Embeddings siameses basados en BERT.	Preciso para búsqueda semántica.	Superado por modelos modernos.
-------	-------------	-------	--------------------------------------	----------------------------------	--------------------------------

Tabla 1 - Comparativa de modelos de embedding

i

Este listado se basa en datos actualizados hasta principios de 2025. Para más detalles, consultar el leaderboard oficial de MTEB en Hugging Face.

Referencias Citadas en este Apéndice

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., & Liu, Z. (2024). *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. arXiv preprint arXiv:2402.03216.

Su, H., Shen, Z., Wang, C., Yu, S., Li, C., & Sun, J. (2022). *One embedder, any task: Instruction-finetuned text embeddings*. arXiv preprint arXiv:2212.09741.

Wang, L., Yang, N., Huang, J., Chang, M. W., & Wang, W. (2022). *Text Embeddings by Weakly-Supervised Contrastive Pre-training*. arXiv preprint arXiv:2212.03533.

(Nota: Las referencias a OpenAI, Cohere, Mistral AI, Nomic AI, Alibaba y otros se basan en la información disponible en sus respectivos sitios web, documentación técnica y model cards en Hugging Face hasta la fecha de corte de este apéndice.)
