Table of Contents

# 6. IMPLEMENTACIÓN

## 6.1 Introducción

Este capítulo detalla la implementación técnica del sistema RAG (Retrieval-Augmented Generation) desarrollado para mejorar la gestión de tickets de soporte técnico mediante recuperación semántica de documentación de Microsoft Azure. La implementación sigue un workflow natural que comienza con la extracción de datos, continúa con el establecimiento de la infraestructura de base de datos vectorial, la generación de embeddings, y culmina con el pipeline de recuperación y generación de respuestas.

La arquitectura técnica adopta principios de ingeniería de software que priorizan la separación de responsabilidades, la extensibilidad y la reproducibilidad científica (McConnell, 2004). El sistema está diseñado para soportar evaluación experimental rigurosa mientras mantiene la flexibilidad necesaria para futuras optimizaciones y expansiones.

## 6.2 Tecnologías Utilizadas

### 6.2.1 Stack Tecnológico Principal

**Lenguaje de Programación:** - Python 3.12.2 como lenguaje principal, seleccionado por su ecosistema maduro en machine learning y procesamiento de lenguaje natural (Van Rossum & Drake, 2009)

**Framework de Interfaz de Usuario:** - Streamlit 1.46.1 para desarrollo rápido de aplicaciones web interactivas con capacidades de visualización de datos (Streamlit Team, 2023)

**Base de Datos Vectorial:** - ChromaDB 0.5.23 como motor de almacenamiento y búsqueda vectorial, seleccionado por su simplicidad operacional y rendimiento en entornos de investigación (ChromaDB Team, 2024)

### 6.2.2 Librerías Especializadas en NLP

**Modelos de Embeddings:**

# Configuración real utilizada en requirements.txt  
sentence-transformers==5.0.0 # Para modelos MPNet, MiniLM, E5-large  
openai==1.93.0 # Para modelo Ada (text-embedding-ada-002)

**Procesamiento de Texto:**

# Librerías para reranking y evaluación  
transformers==4.44.0 # Para CrossEncoder ms-marco-MiniLM-L-6-v2  
torch==2.2.2 # Backend para modelos PyTorch  
bert-score==0.3.13 # Para métricas de evaluación semántica

### 6.2.3 Infraestructura de Evaluación

**Entorno de Cómputo:** - Google Colab con GPU Tesla T4 para aceleración de cómputo en evaluaciones masivas - Jupyter Notebooks para prototipado y análisis exploratorio - Local execution con CPU Intel Core i7 y 16GB RAM para desarrollo iterativo

**Almacenamiento de Datos:** - Formato Parquet para almacenamiento eficiente de embeddings pre-computados - JSON para metadatos y resultados de evaluación - Google Drive para sincronización automática de resultados experimentales

## 6.3 Extracción Automatizada de Datos desde Microsoft Learn

### 6.3.1 Herramientas y Técnicas de Web Scraping

La extracción de datos representa la primera fase del proyecto y constituye la base fundamental que alimenta todo el sistema RAG. La implementación combina Selenium para navegación dinámica y BeautifulSoup para parsing de contenido, estableciendo una metodología robusta para la recolección de datos técnicos especializados.

**Arquitectura de Scraping:** - Selenium WebDriver con ChromeDriver para manejo de JavaScript y contenido dinámico - BeautifulSoup 4 para parsing estructurado de HTML renderizado - Estrategias de espera adaptativa para carga asíncrona de contenido - Manejo robusto de errores con reintentos automáticos

{El código específico de scraping requiere verificación de archivos originales para proporcionar implementación real utilizada}

**Desafíos Técnicos Identificados:** - Carga asíncrona del contenido en Microsoft Learn requirió WebDriverWait con condiciones específicas - Estructura HTML variable entre páginas necesitó selectores CSS robustos - Volumen de datos (>20,000 preguntas) requirió sistema incremental con checkpoints

### 6.3.2 Proceso de Extracción de Documentación

El proceso de extracción de documentación técnica de Microsoft Learn sigue una metodología estructurada que garantiza la completitud y calidad de los datos recolectados:

**Pipeline de Extracción Documentada:**

1. **Identificación de Puntos de Entrada:** Navegación desde índices principales de Azure (https://learn.microsoft.com/en-us/azure/)
2. **Crawling Recursivo:** Seguimiento de enlaces internos con filtrado de relevancia
3. **Extracción de Contenido:** Parsing de elementos estructurales específicos (títulos, contenido, metadatos)
4. **Normalización de Datos:** Limpieza de HTML, normalización de URLs, y estructuración JSON

**Estructura de Datos Implementada:**

{  
 "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]",  
 "related\_links": ["https://learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml", ...]  
}

**Resultados de Extracción Verificados:** - **62,417 documentos únicos** relacionados con Azure - **187,031 chunks procesables** después de segmentación - Cobertura completa de servicios principales de Azure - Metadatos ricos incluyendo títulos, URLs, y contenido textual

### 6.3.3 Proceso de Extracción de Preguntas y Respuestas

La extracción de preguntas desde Microsoft Q&A implementa técnicas especializadas para capturar no solo el contenido textual sino también las relaciones semánticas y la validación comunitaria:

**Metodología de Extracción Q&A:**

1. **Navegación Sistemática:** Recorrido de páginas indexadas bajo el tag “Azure”
2. **Extracción de Metadatos:** Captura de fecha, etiquetas, y métricas de interacción
3. **Identificación de Respuestas Aceptadas:** Filtrado de respuestas validadas por la comunidad
4. **Extracción de Enlaces:** Parsing de URLs a documentación oficial en respuestas

**Estructura de Datos Q&A:**

{  
 "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"  
}

**Dataset Resultante Verificado:** - **13,436 preguntas técnicas** con contenido completo - **2,067 preguntas con enlaces validados** a documentación oficial (ground truth) - Distribución temporal concentrada en 2023-2024 (77.3% del total) - Longitud promedio de pregunta: 119.9 tokens - Longitud promedio de respuesta: 221.6 tokens

### 6.3.4 Consideraciones Éticas y Legales del Uso de Documentación Técnica

#### 6.3.4.1 Marco Legal y Licenciamiento

El uso de documentación de Microsoft Learn se fundamenta en el cumplimiento estricto de las condiciones de licenciamiento establecidas por Microsoft Corporation. La documentación técnica disponible en learn.microsoft.com se encuentra licenciada bajo Creative Commons Attribution 4.0 International (CC BY 4.0), excepto donde se indique lo contrario (Microsoft Corporation, 2024).

**Condiciones de Uso Aplicadas:** - **Atribución Completa:** Reconocimiento explícito de Microsoft como autor del material original - **Uso Académico:** Aplicación exclusiva para fines de investigación y educación superior - **No Redistribución:** Ausencia de publicación o exposición del contenido textual íntegro - **Transformación Académica:** Uso como insumo para modelos de recuperación semántica

#### 6.3.4.2 Buenas Prácticas Implementadas

**Respeto por Recursos del Servidor:** - Implementación de delays adaptativos entre requests para evitar sobrecarga - Respeto por directivas robots.txt y headers de rate limiting - Uso de User-Agent identificativo para transparencia de propósito académico

**Protección de Datos y Privacidad:** - Exclusión de información personal o identificadores de usuarios - Anonimización de metadatos no esenciales para la investigación - Almacenamiento seguro con acceso restringido a datos recolectados

**Transparencia y Reproducibilidad:** - Documentación completa de fuentes y metodologías de extracción - Mantenimiento de trazabilidad mediante URLs originales - Disponibilidad de scripts y procedimientos para validación independiente

#### 6.3.4.3 Limitaciones y Salvaguardas

**Limitaciones Voluntariamente Adoptadas:** - Exclusión de contenido marcado como confidencial o beta - Respeto por contenido con restricciones específicas de licenciamiento - Limitación temporal de datos para evitar obsolescencia de información

**Salvaguardas Implementadas:** - Monitoreo regular de cambios en términos de uso - Procedimientos de eliminación de datos si requerido por el propietario - Contacto establecido con Microsoft para transparencia de investigación

La implementación de estas consideraciones éticas asegura que el proyecto mantiene los más altos estándares de integridad académica mientras contribuye al avance del conocimiento en sistemas de recuperación de información técnica.

**Nota sobre Implementación de Scraping:** El código específico de scraping no se incluye en el sistema actual debido a que la extracción de datos se realizó en una fase previa del proyecto. Los datos extraídos se almacenaron en formato estructurado (JSON y Parquet) y se utilizan directamente desde ChromaDB en la implementación actual.

## 6.4 Implementación de ChromaDB

### 6.4.1 Arquitectura de Base de Datos Vectorial

Una vez completada la extracción de datos, el siguiente paso fue establecer la infraestructura de almacenamiento vectorial. ChromaDB fue seleccionado como base de datos vectorial principal después de una migración desde Weaviate, basada en criterios de optimización para flujos de investigación académica.

**Justificación de Migración Weaviate → ChromaDB:**

**Weaviate (implementación inicial):** - Ventajas: Escalabilidad empresarial, API GraphQL, módulos especializados - Limitaciones: Latencia de red (150-300ms por consulta), dependencia de conectividad externa - Aplicabilidad: Óptimo para aplicaciones de producción distribuida

**ChromaDB (implementación final):** - Ventajas: Latencia local (<10ms), portabilidad de datos, simplicidad de configuración - Aplicabilidad: Óptimo para investigación y desarrollo iterativo

### 6.4.2 Configuración e Inicialización

La configuración de ChromaDB implementa un patrón de cliente singleton con manejo de conexiones persistentes, optimizado para el patrón de uso académico:

# Implementación real en src/services/storage/chromadb\_utils.py  
class ChromaDBClientWrapper:  
 """Wrapper singleton para cliente ChromaDB con configuración optimizada."""  
   
 def \_\_init\_\_(self, chromadb\_path: str = "/Users/haroldgomez/chromadb2"):  
 """Inicialización con path absoluto para consistencia."""  
 self.chromadb\_path = chromadb\_path  
 self.\_client = None  
 self.\_collections = {}  
   
 @property  
 def client(self):  
 """Cliente lazy-loaded con configuración persistente."""  
 if self.\_client is None:  
 self.\_client = chromadb.PersistentClient(path=self.chromadb\_path)  
 return self.\_client  
   
 def get\_collection(self, collection\_name: str):  
 """Acceso cached a colecciones con validación de existencia."""  
 if collection\_name not in self.\_collections:  
 try:  
 self.\_collections[collection\_name] = self.client.get\_collection(collection\_name)  
 except chromadb.errors.InvalidCollectionException:  
 logger.error(f"Colección {collection\_name} no encontrada")  
 return None  
 return self.\_collections[collection\_name]

### 6.4.3 Gestión de Colecciones Multi-Modelo

La arquitectura de almacenamiento implementa colecciones separadas para cada modelo de embedding, permitiendo comparaciones directas sin interferencia cruzada:

# Configuración real de src/config/config.py  
CHROMADB\_COLLECTION\_CONFIG = {  
 "multi-qa-mpnet-base-dot-v1": {  
 "documents": "docs\_mpnet", # 187,031 documentos - 768D  
 "questions": "questions\_mpnet", # 13,436 preguntas - 768D  
 "questions\_withlinks": "questions\_withlinks" # 2,067 preguntas validadas  
 },  
 "all-MiniLM-L6-v2": {  
 "documents": "docs\_minilm", # 187,031 documentos - 384D  
 "questions": "questions\_minilm", # 13,436 preguntas - 384D  
 "questions\_withlinks": "questions\_withlinks"  
 },  
 "ada": {  
 "documents": "docs\_ada", # 187,031 documentos - 1536D  
 "questions": "questions\_ada", # 13,436 preguntas - 1536D  
 "questions\_withlinks": "questions\_withlinks"  
 },  
 "e5-large-v2": {  
 "documents": "docs\_e5large", # 187,031 documentos - 1024D  
 "questions": "questions\_e5large", # 13,436 preguntas - 1024D  
 "questions\_withlinks": "questions\_withlinks"  
 }  
}

### 6.4.4 Optimizaciones de Rendimiento

**Almacenamiento Eficiente:** - Utilización de formato Parquet para embeddings pre-computados - Compresión adaptativa basada en dimensionalidad de vectores - Indexación optimizada para consultas de similitud coseno

**Gestión de Memoria:** - Carga lazy de colecciones para minimizar footprint de memoria - Cached de resultados frecuentes con LRU eviction - Batch processing para operaciones masivas

**Métricas de Rendimiento Observadas:** - Latencia de consulta promedio: <10ms para top-k=10 - Throughput: ~241 documentos/segundo para embedding generation - Almacenamiento total: 6.48 GB para todas las colecciones

## 6.5 Arquitectura del Sistema RAG

### 6.5.1 Componente de Indexación y Embeddings

Tras establecer la infraestructura de ChromaDB, el siguiente paso fue implementar la generación y gestión de embeddings múltiples. El sistema permite comparación directa entre diferentes modelos de representación vectorial:

# Implementación real en src/data/embedding.py  
class EmbeddingClient:  
 def \_\_init\_\_(self,   
 model\_name: str = "sentence-transformers/all-MiniLM-L6-v2",   
 document\_model\_name: str = "sentence-transformers/multi-qa-mpnet-base-dot-v1",  
 huggingface\_api\_key: str | None = None):  
 """  
 Initialize embedding client with lazy loading to prevent memory issues.  
   
 Args:  
 model\_name: Model for queries (default: MiniLM for questions)  
 document\_model\_name: Model for documents (default: MPNet for documents)  
 huggingface\_api\_key: Optional HuggingFace API key  
 """  
 # Set HuggingFace token before importing SentenceTransformer  
 if huggingface\_api\_key:  
 os.environ["HUGGINGFACE\_HUB\_TOKEN"] = huggingface\_api\_key  
   
 # Store model names for lazy loading  
 self.model\_name = model\_name  
 self.document\_model\_name = document\_model\_name  
   
 # Initialize models as None - will be loaded on first use  
 self.\_query\_model = None  
 self.\_document\_model = None  
 self.\_model\_lock = threading.Lock()  
   
 def generate\_query\_embedding(self, text: str) -> List[float]:  
 """Generate embedding using query model (MiniLM) - for questions."""  
 return self.generate\_embedding(text, use\_document\_model=False)  
   
 def generate\_document\_embedding(self, text: str) -> List[float]:  
 """Generate embedding using document model (MPNet) - for documents."""  
 return self.generate\_embedding(text, use\_document\_model=True)

### 6.5.2 Componente de Búsqueda Vectorial

#### 6.5.2.1 Búsqueda Vectorial con Filtrado de Diversidad

El componente de búsqueda implementa algoritmos de similitud coseno con filtrado de diversidad para evitar resultados redundantes:

# Implementación en src/services/storage/chromadb\_utils.py  
def search\_docs\_by\_vector(self, vector: np.ndarray, top\_k: int = 10,   
 diversity\_threshold: float = 0.85) -> List[Dict]:  
 """Búsqueda vectorial con filtrado de diversidad semántica."""  
   
 # Búsqueda inicial con sobremuestreo para filtrado posterior  
 fetch\_limit = min(top\_k \* 3, 50) # Balancear calidad vs rendimiento  
   
 results = self.\_docs\_collection.query(  
 query\_embeddings=[vector.tolist()],  
 n\_results=fetch\_limit,  
 include=['embeddings', 'metadatas', 'documents', 'distances']  
 )  
   
 # Conversión a objetos estructurados  
 objects = self.\_format\_search\_results(results)  
   
 # Aplicar filtrado de diversidad  
 return self.\_apply\_diversity\_filtering(objects, top\_k, diversity\_threshold)  
  
def \_apply\_diversity\_filtering(self, docs: List[Dict], top\_k: int,   
 threshold: float) -> List[Dict]:  
 """Filtrado de diversidad para evitar documentos semánticamente redundantes."""  
 selected = []  
   
 for doc in docs:  
 is\_diverse = True  
 doc\_embedding = np.array(doc['embedding'])  
   
 for selected\_doc in selected:  
 selected\_embedding = np.array(selected\_doc['embedding'])  
 similarity = cosine\_similarity(  
 doc\_embedding.reshape(1, -1),  
 selected\_embedding.reshape(1, -1)  
 )[0][0]  
   
 if similarity > threshold:  
 is\_diverse = False  
 break  
   
 if is\_diverse:  
 selected.append(doc)  
 if len(selected) >= top\_k:  
 break  
   
 return selected

#### 6.5.2.2 Búsqueda Híbrida por Enlaces Validados

El sistema implementa búsqueda híbrida que combina recuperación por enlaces directos con búsqueda vectorial:

def lookup\_docs\_by\_links\_batch(self, links: List[str], batch\_size: int = 50) -> List[Dict]:  
 """Búsqueda batch optimizada por enlaces con normalización URL."""  
   
 # Normalización de URLs para coincidencia robusta  
 normalized\_links = [normalize\_url(link) for link in links if link]  
   
 found\_docs = []  
 for i in range(0, len(normalized\_links), batch\_size):  
 link\_batch = normalized\_links[i:i + batch\_size]  
   
 # Consulta ChromaDB con límite de rendimiento (5000 docs)  
 results = self.\_docs\_collection.query(  
 query\_texts=[""], # Query dummy para obtener todos  
 n\_results=5000, # Límite para mantener rendimiento  
 include=['metadatas', 'documents']  
 )  
   
 # Filtrado por enlaces normalizados  
 for j, metadata in enumerate(results['metadatas'][0]):  
 doc\_link = normalize\_url(metadata.get('link', ''))  
 if doc\_link in link\_batch:  
 found\_docs.append({  
 'link': doc\_link,  
 'title': metadata.get('title', ''),  
 'content': results['documents'][0][j],  
 'source': 'link\_lookup'  
 })  
   
 return found\_docs

### 6.5.3 Componente de Evaluación

La implementación de métricas sigue estándares establecidos en literatura de recuperación de información:

# Implementación en src/evaluation/metrics/retrieval.py  
def calculate\_retrieval\_metrics(retrieved\_docs: List[Dict],   
 ground\_truth\_links: Set[str],  
 k\_values: List[int] = [1, 3, 5, 10, 15]) -> Dict[str, float]:  
 """Cálculo comprehensivo de métricas de recuperación."""  
   
 metrics = {}  
   
 # Normalización de enlaces para comparación robusta  
 retrieved\_links = [normalize\_url(doc.get('link', '')) for doc in retrieved\_docs]  
 normalized\_ground\_truth = {normalize\_url(link) for link in ground\_truth\_links}  
   
 # Mean Reciprocal Rank  
 metrics['MRR'] = calculate\_mrr(retrieved\_links, normalized\_ground\_truth)  
   
 # Métricas @k para diferentes valores de k  
 for k in k\_values:  
 precision\_k = calculate\_precision\_at\_k(retrieved\_links, normalized\_ground\_truth, k)  
 recall\_k = calculate\_recall\_at\_k(retrieved\_links, normalized\_ground\_truth, k)  
   
 metrics[f'Precision@{k}'] = precision\_k  
 metrics[f'Recall@{k}'] = recall\_k  
 metrics[f'F1@{k}'] = calculate\_f1\_score(precision\_k, recall\_k)  
 metrics[f'NDCG@{k}'] = calculate\_ndcg\_at\_k(retrieved\_links, normalized\_ground\_truth, k)  
 metrics[f'MAP@{k}'] = calculate\_map\_at\_k(retrieved\_links, normalized\_ground\_truth, k)  
   
 return metrics

## 6.6 Pipeline de Procesamiento RAG

### 6.6.1 Pipeline End-to-End

El pipeline de procesamiento implementa una arquitectura multi-etapa que integra todos los componentes desarrollados previamente:

# Implementación principal en src/core/qa\_pipeline.py  
# Función principal simplificada - la implementación completa incluye múltiples parámetros  
# para diferentes modelos generativos (local, OpenRouter, Gemini)  
def answer\_question\_with\_rag(question: str, chromadb\_wrapper: ChromaDBClientWrapper,  
 embedding\_client: EmbeddingClient, \*\*kwargs) -> Dict:  
 """Pipeline RAG completo con logs detallados y métricas."""  
   
 pipeline\_start = time.time()  
 log = []  
   
 # Etapa 1: Query Refinement y Preparación  
 log.append("1. Iniciando refinamiento de consulta")  
 refined\_query, refinement\_log = refine\_and\_prepare\_query(question, embedding\_client)  
 log.extend(refinement\_log)  
   
 # Etapa 2: Generación de Embedding de Consulta  
 log.append("2. Generando embedding de consulta")  
 query\_vector = embedding\_client.generate\_query\_embedding(refined\_query, model\_name)  
   
 # Etapa 3: Búsqueda de Preguntas Similares  
 log.append("3. Buscando preguntas similares")  
 similar\_questions = chromadb\_wrapper.search\_questions\_by\_vector(  
 query\_vector, model\_name, top\_k=30  
 )  
   
 # Etapa 4: Extracción de Enlaces desde Respuestas  
 all\_links = []  
 for q in similar\_questions[:5]: # Top-5 preguntas más similares  
 accepted\_answer = q.get('accepted\_answer', '')  
 if accepted\_answer:  
 extracted\_links = extract\_urls\_from\_answer(accepted\_answer)  
 all\_links.extend(extracted\_links)  
   
 log.append(f"4. Extraídos {len(all\_links)} enlaces de respuestas")  
   
 # Etapa 5: Recuperación Híbrida de Documentos  
 log.append("5. Iniciando recuperación híbrida de documentos")  
   
 # 5a. Búsqueda por enlaces directos  
 linked\_docs = []  
 if all\_links:  
 linked\_docs = chromadb\_wrapper.lookup\_docs\_by\_links\_batch(all\_links)  
 log.append(f" - Encontrados {len(linked\_docs)} documentos por enlaces")  
   
 # 5b. Búsqueda vectorial de documentos  
 document\_vector = embedding\_client.generate\_document\_embedding(refined\_query, model\_name)  
 vector\_docs = chromadb\_wrapper.search\_docs\_by\_vector(  
 document\_vector, model\_name, top\_k=20, diversity\_threshold=0.85  
 )  
 log.append(f" - Encontrados {len(vector\_docs)} documentos por similitud vectorial")  
   
 # Etapa 6: Deduplicación y Fusión  
 unique\_docs = deduplicate\_documents(linked\_docs + vector\_docs)  
 log.append(f"6. Documentos únicos después de deduplicación: {len(unique\_docs)}")  
   
 # Etapa 7: Reranking Neural (Opcional)  
 final\_docs = unique\_docs  
 if use\_reranking and len(unique\_docs) > 1:  
 log.append("7. Aplicando reranking con CrossEncoder")  
 final\_docs = rerank\_with\_llm(question, unique\_docs, openai\_client, top\_k=top\_k)  
 log.append(f" - Documentos después del reranking: {len(final\_docs)}")  
   
 # Etapa 8: Generación de Respuesta  
 log.append("8. Generando respuesta final")  
 generated\_answer = generate\_rag\_answer(question, final\_docs[:3])  
   
 pipeline\_time = time.time() - pipeline\_start  
 log.append(f"Pipeline completado en {pipeline\_time:.2f} segundos")  
   
 return {  
 'question': question,  
 'answer': generated\_answer,  
 'retrieved\_docs': final\_docs,  
 'similar\_questions': similar\_questions,  
 'processing\_log': log,  
 'metrics': {  
 'total\_time': pipeline\_time,  
 'documents\_retrieved': len(final\_docs),  
 'similar\_questions\_found': len(similar\_questions)  
 }  
 }

### 6.6.2 Reranking con CrossEncoder

El componente de reranking implementa el modelo ms-marco-MiniLM-L-6-v2 con normalización Min-Max:

# Implementación real en src/core/reranker.py  
def rerank\_with\_llm(question: str, docs: List[dict], openai\_client: OpenAI,   
 top\_k: int = 10, embedding\_model: str = None) -> List[dict]:  
 """  
 Reranks documents using a local CrossEncoder model with sigmoid normalization.  
   
 Uses sigmoid normalization instead of softmax to ensure scores are comparable  
 across different embedding models regardless of the number of documents returned.  
 """  
 if not docs:  
 return []  
  
 # The CrossEncoder model expects pairs of [query, passage]  
 model\_inputs = [[question, doc.get("content", "")] for doc in docs]  
   
 # Initialize a light-weight, fast, and effective cross-encoder  
 cross\_encoder = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2', max\_length=512)  
   
 # Predict the raw logit scores  
 raw\_scores = cross\_encoder.predict(model\_inputs)  
   
 # Apply sigmoid normalization to CrossEncoder scores  
 try:  
 raw\_scores = np.array(raw\_scores)  
 # Apply sigmoid: 1 / (1 + e^(-x))  
 # This maps CrossEncoder logits to [0,1] probabilities  
 final\_scores = 1 / (1 + np.exp(-raw\_scores))  
 except (OverflowError, ZeroDivisionError):  
 # Fallback: Min-max normalization if sigmoid fails  
 raw\_scores = np.array(raw\_scores)  
 min\_score = np.min(raw\_scores)  
 max\_score = np.max(raw\_scores)  
 if max\_score > min\_score:  
 final\_scores = (raw\_scores - min\_score) / (max\_score - min\_score)  
 else:  
 final\_scores = np.ones\_like(raw\_scores) \* 0.5  
 print(f"[WARNING] Sigmoid normalization failed, using min-max normalization")  
  
 # Add final scores to the documents  
 for doc, score in zip(docs, final\_scores):  
 if "score" in doc and "pre\_rerank\_score" not in doc:  
 doc["pre\_rerank\_score"] = doc["score"]  
 doc["score"] = float(score)  
   
 # Sort documents by the new score in descending order  
 return sorted(docs, key=lambda d: d.get("score", 0.0), reverse=True)[:top\_k]

### 6.6.3 Generación de Respuestas Multi-Modal

El sistema soporta múltiples backends de generación de respuestas:

# Implementación en src/services/answer\_generation/local.py  
def generate\_final\_answer\_local(question: str, context\_docs: List[Dict],   
 model\_name: str = "TinyLlama-1.1B") -> str:  
 """Generación de respuesta con modelos locales."""  
   
 # Preparación de contexto optimizado  
 context\_text = "\n\n".join([  
 f"Document {i+1}: {doc.get('title', 'Untitled')}\n{doc.get('content', '')[:800]}"  
 for i, doc in enumerate(context\_docs[:3])  
 ])  
   
 prompt = f"""Based on the following context documents, answer the question accurately and concisely.  
  
Context:  
{context\_text}  
  
Question: {question}  
  
Answer:"""  
   
 # Generación con modelo local  
 local\_client = get\_local\_model\_client(model\_name)  
   
 response = local\_client.chat.completions.create(  
 model=model\_name,  
 messages=[{"role": "user", "content": prompt}],  
 max\_tokens=200,  
 temperature=0.1,  
 stream=False  
 )  
   
 return response.choices[0].message.content.strip()

## 6.7 Interfaz de Usuario (Streamlit)

### 6.7.1 Arquitectura Multi-Página

La interfaz de usuario implementa una aplicación Streamlit multi-página que integra todos los componentes del sistema:

# Implementación principal en src/apps/main\_qa\_app.py  
def main():  
 """Aplicación principal con navegación multi-página."""  
   
 st.set\_page\_config(  
 page\_title="Sistema RAG - Soporte Técnico Azure",  
 page\_icon="🔍",  
 layout="wide",  
 initial\_sidebar\_state="expanded"  
 )  
   
 # Sidebar con navegación  
 with st.sidebar:  
 st.title("🔍 Sistema RAG")  
 st.markdown("---")  
   
 page = st.selectbox(  
 "Selecciona una página:",  
 ["🤖 Consulta Q&A", "📊 Métricas Cumulativas", "⚙️ Configuración"]  
 )  
   
 # Enrutamiento de páginas  
 if page == "🤖 Consulta Q&A":  
 render\_qa\_interface()  
 elif page == "📊 Métricas Cumulativas":  
 render\_metrics\_dashboard()  
 elif page == "⚙️ Configuración":  
 render\_configuration\_panel()

### 6.7.2 Interfaz de Consulta Q&A

def render\_qa\_interface():  
 """Interfaz principal de consulta Q&A con RAG."""  
   
 st.title("Sistema de Consulta Q&A con Recuperación Semántica")  
   
 # Configuración en columnas  
 col1, col2, col3 = st.columns([2, 1, 1])  
   
 with col1:  
 question = st.text\_area(  
 "Escribe tu pregunta sobre Azure:",  
 height=100,  
 placeholder="Ejemplo: ¿Cómo configurar Azure Active Directory para autenticación?"  
 )  
   
 with col2:  
 model\_name = st.selectbox(  
 "Modelo de Embedding:",  
 ['mpnet', 'ada', 'minilm', 'e5large'],  
 index=0  
 )  
   
 top\_k = st.slider("Top-K Documentos:", 5, 20, 15)  
   
 with col3:  
 use\_reranking = st.checkbox("Usar CrossEncoder", value=True)  
 show\_sources = st.checkbox("Mostrar Fuentes", value=True)  
   
 if st.button("🔍 Buscar Respuesta", type="primary"):  
 if question.strip():  
 with st.spinner("Procesando consulta..."):  
 # Ejecución del pipeline RAG  
 result = answer\_question\_with\_rag(  
 question=question,  
 chromadb\_wrapper=get\_chromadb\_wrapper(),  
 embedding\_client=get\_embedding\_client(),  
 model\_name=model\_name,  
 top\_k=top\_k,  
 use\_reranking=use\_reranking  
 )  
   
 # Renderizado de resultados  
 render\_qa\_results(result, show\_sources)

### 6.7.3 Dashboard de Métricas

def render\_metrics\_dashboard():  
 """Dashboard de métricas experimentales con visualizaciones."""  
   
 st.title("📊 Resultados de Evaluación Experimental")  
   
 # Carga de resultados experimentales  
 results\_file = st.selectbox(  
 "Selecciona archivo de resultados:",  
 get\_available\_results\_files()  
 )  
   
 if results\_file:  
 data = load\_experimental\_results(results\_file)  
   
 # Información general del experimento  
 col1, col2, col3, col4 = st.columns(4)  
 with col1:  
 st.metric("Preguntas Evaluadas", data['config']['num\_questions'])  
 with col2:  
 st.metric("Modelos Comparados", data['config']['models\_evaluated'])  
 with col3:  
 st.metric("Top-K", data['config']['top\_k'])  
 with col4:  
 st.metric("Método Reranking", data['config']['reranking\_method'])  
   
 # Visualización comparativa de modelos  
 render\_model\_comparison\_charts(data)  
   
 # Tabla de métricas detalladas  
 st.subheader("📋 Métricas Detalladas")  
 metrics\_df = create\_metrics\_comparison\_table(data)  
 st.dataframe(metrics\_df, use\_container\_width=True)

## 6.8 Optimizaciones y Mejoras

### 6.8.1 Optimizaciones de Rendimiento

**Caching Inteligente:** - Implementación de LRU cache para modelos de embeddings cargados - Cache persistente de resultados de consultas frecuentes - Lazy loading de componentes pesados (CrossEncoder, modelos locales)

**Batch Processing:** - Procesamiento en lotes para búsquedas por enlaces (batch\_size=50) - Vectorización batch para generación masiva de embeddings - Paralelización de evaluaciones experimentales

**Gestión de Memoria:** - Liberación automática de memoria después de evaluaciones grandes - Uso de generators para procesamiento de datasets extensos - Monitoreo activo de uso de memoria con alertas

### 6.8.2 Mejoras de Calidad

**Filtrado de Diversidad:** - Algoritmo de diversidad semántica para evitar documentos redundantes - Threshold adaptativo basado en distribución de similitudes - Preservación de documentos altamente relevantes independiente de diversidad

**Normalización Robusta:** - Normalización de URLs para matching preciso entre enlaces - Limpieza de texto adaptativa para diferentes fuentes - Manejo consistente de encoding y caracteres especiales

**Validación de Calidad:** - Verificación automática de integridad de embeddings - Detección de documentos corrompidos o incompletos - Métricas de calidad de datos integradas en pipeline

### 6.8.3 Extensibilidad Arquitectónica

**Interfaces Modulares:** - Separación clara entre capas de datos, lógica y presentación - Interfaces estándar para incorporación de nuevos modelos - Plugin architecture para métricas de evaluación customizadas

**Configuración Flexible:** - Archivos de configuración JSON para parámetros del sistema - Variables de entorno para secrets y paths - Override dinámico de configuraciones via interfaz web

**Logging y Monitoreo:** - Logging estructurado con niveles configurables - Métricas de rendimiento integradas - Trazabilidad completa de requests y resultados

La implementación técnica descrita sigue el workflow natural del proyecto: desde la extracción inicial de datos, pasando por el establecimiento de la infraestructura de base de datos vectorial, la generación de embeddings, hasta culminar en un pipeline RAG completo con interfaz de usuario comprehensiva. Esta arquitectura modular y las optimizaciones implementadas proporcionan una base sólida tanto para investigación académica como para potencial implementación en entornos de producción.

## 6.9 Referencias del Capítulo

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0 step-by-step data mining guide. SPSS Inc.

ChromaDB Team. (2024). ChromaDB: The AI-native open-source embedding database. https://www.trychroma.com/

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction (2nd ed.). Microsoft Press.

Microsoft Corporation. (2024). Microsoft Learn Terms of Use. https://learn.microsoft.com/en-us/legal/

Streamlit Team. (2023). Streamlit: The fastest way to build and share data apps. https://streamlit.io/

Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace Independent Publishing Platform.