Table of Contents

# 3. MARCO TEÓRICO

## 3.1 Introducción

El marco teórico de este proyecto se fundamenta en la convergencia de múltiples dominios tecnológicos que han experimentado avances significativos en la última década: recuperación de información semántica, modelos de embeddings densos, arquitecturas de Retrieval-Augmented Generation (RAG), y bases de datos vectoriales optimizadas. Esta convergencia ha habilitado el desarrollo de sistemas de soporte técnico automatizado que superan las limitaciones de los enfoques tradicionales basados en coincidencia léxica.

La naturaleza técnica y especializada del dominio de Microsoft Azure presenta desafíos únicos que requieren una comprensión profunda de los fundamentos teóricos que sustentan las tecnologías empleadas. Este capítulo establece los cimientos conceptuales necesarios para comprender la arquitectura, implementación y evaluación del sistema RAG desarrollado, proporcionando un análisis exhaustivo de cada componente tecnológico y su contribución al objetivo general de automatización inteligente del soporte técnico.

## 3.2 Fundamentos de Recuperación de Información

### 3.2.1 Evolución de los Paradigmas de Recuperación

La recuperación de información (Information Retrieval, IR) ha evolucionado desde modelos probabilísticos clásicos hacia enfoques semánticos basados en representaciones vectoriales densas. El modelo vectorial tradicional, introducido por Salton et al. (1975), representa documentos y consultas como vectores en un espacio multidimensional donde cada dimensión corresponde a un término del vocabulario. Sin embargo, este enfoque sufre de limitaciones fundamentales relacionadas con la maldición de la dimensionalidad y la incapacidad de capturar relaciones semánticas implícitas.

El cambio paradigmático hacia recuperación semántica densa ha sido posible gracias al desarrollo de modelos de lenguaje preentrenados capaces de generar representaciones vectoriales que preservan información semántica y contextual (Karpukhin et al., 2020). Estos modelos transforman texto en vectores de baja dimensión (típicamente 256-1536 dimensiones) que capturan similitudes semánticas no evidentes en el nivel léxico.

### 3.2.2 Fundamentos Matemáticos de Similitud Semántica

La similitud semántica en espacios vectoriales densos se cuantifica típicamente mediante la similitud coseno, definida como:

sim(q, d) = (q · d) / (||q|| × ||d||)

donde q representa el vector de consulta, d el vector de documento, y || || denota la norma euclidiana. Esta métrica normaliza los vectores, enfocándose en la orientación angular más que en la magnitud, lo que resulta especialmente apropiado para comparaciones semánticas donde la longitud del documento es menos relevante que su contenido conceptual.

La efectividad de esta aproximación depende críticamente de la calidad de las representaciones vectoriales, que deben preservar relaciones semánticas de manera que documentos conceptualmente similares mantengan proximidad en el espacio vectorial (Reimers & Gurevych, 2019).

### 3.2.3 Arquitecturas de Recuperación Multi-Etapa

Los sistemas modernos de recuperación implementan arquitecturas multi-etapa que optimizan el balance entre recall y precisión mediante un proceso de refinamiento progresivo (Qu et al., 2021). El pipeline típico incluye:

1. **Recuperación inicial (Dense Retrieval)**: Búsqueda vectorial eficiente sobre el corpus completo utilizando similitud coseno
2. **Reranking (CrossEncoder)**: Refinamiento de candidatos utilizando modelos más sofisticados que procesan conjuntamente consulta y documento
3. **Filtrado final**: Aplicación de reglas de negocio y thresholding para optimizar precisión

Esta arquitectura permite escalar a corpus de gran tamaño manteniendo alta precisión en los resultados finales.

## 3.3 Modelos de Embeddings

### 3.3.1 OpenAI Ada (text-embedding-ada-002)

#### 3.3.1.1 Arquitectura y Características Técnicas

OpenAI Ada representa el estado del arte en modelos de embeddings comerciales, implementando una arquitectura Transformer optimizada específicamente para generación de representaciones vectoriales densas (OpenAI, 2023). El modelo genera vectores de 1,536 dimensiones optimizados para tareas de similitud semántica y recuperación de información.

**Características técnicas verificadas:** - **Dimensionalidad**: 1,536 dimensiones por vector - **Longitud máxima de contexto**: 8,191 tokens - **Arquitectura**: Transformer con optimizaciones propietarias para embeddings - **Normalización**: Vectores unitarios (norma L2 = 1.0)

#### 3.3.1.2 Optimizaciones para Recuperación Semántica

Ada incorpora técnicas avanzadas de preentrenamiento contrastivo que optimizan la representación vectorial para tareas de similitud semántica. El modelo ha sido entrenado en un corpus diverso que incluye documentación técnica, lo que resulta particularmente beneficioso para dominios especializados como Microsoft Azure.

# Implementación real utilizada en el proyecto  
client = OpenAI(api\_key=os.getenv('OPENAI\_API\_KEY'))  
response = client.embeddings.create(  
 input=question,  
 model="text-embedding-ada-002"  
)  
ada\_embedding = np.array(response.data[0].embedding)

#### 3.3.1.3 Limitaciones y Consideraciones

Ada presenta limitaciones relacionadas con su naturaleza propietaria, incluyendo dependencia de API externa, costos operacionales variables, y opacidad arquitectónica que impide optimizaciones específicas del dominio. Además, su rendimiento puede degradarse en terminología altamente especializada no representada en el corpus de entrenamiento.

### 3.3.2 Sentence-BERT (MPNet, MiniLM)

#### 3.3.2.1 MPNet: Arquitectura de Preentrenamiento Híbrida

MPNet (Masked and Permuted Pre-training) combina las ventajas de BERT y XLNet mediante una estrategia de preentrenamiento que incorpora tanto masked language modeling como permuted language modeling (Song et al., 2020). Esta aproximación híbrida resulta en representaciones más robustas para tareas de recuperación semántica.

**Especificaciones técnicas del modelo multi-qa-mpnet-base-dot-v1:** - **Dimensionalidad**: 768 dimensiones - **Arquitectura**: 12 capas transformer con 12 cabezas de atención - **Parámetros**: ~110M parámetros - **Especialización**: Fine-tuning específico en pares pregunta-respuesta

# Implementación real con prefijo optimizado  
if model\_name == 'mpnet':  
 prefixed\_question = f"query: {question}"  
 embedding = self.models[model\_name].encode(prefixed\_question)

El prefijo “query:” es crucial para el rendimiento óptimo de MPNet, ya que el modelo fue entrenado con esta convención para distinguir entre consultas y documentos durante el fine-tuning.

#### 3.3.2.2 MiniLM: Optimización Eficiencia-Rendimiento

MiniLM implementa destilación de conocimiento desde modelos BERT más grandes, manteniendo calidad semántica mientras reduce significativamente los requerimientos computacionales (Wang et al., 2020). Esta optimización es particularmente valiosa en aplicaciones de producción con restricciones de recursos.

**Especificaciones del modelo all-MiniLM-L6-v2:** - **Dimensionalidad**: 384 dimensiones - **Capas**: 6 capas transformer - **Parámetros**: ~22M parámetros - **Velocidad**: ~5x más rápido que BERT-base

La reducción dimensional y arquitectónica se compensa mediante técnicas avanzadas de destilación que preservan información semántica crítica en el espacio vectorial de menor dimensión.

### 3.3.3 E5-Large: Embeddings Especializados en Recuperación

#### 3.3.3.1 Arquitectura de Preentrenamiento Contrastivo

E5-Large (Embeddings from bidirectional Encoder representations) implementa una estrategia de preentrenamiento contrastivo específicamente optimizada para tareas de recuperación de información (Wang et al., 2022). El modelo utiliza técnicas de aprendizaje auto-supervisado que maximizan la similitud entre pares relacionados mientras minimizan la similitud entre pares no relacionados.

**Características técnicas:** - **Dimensionalidad**: 1,024 dimensiones - **Arquitectura**: Transformer con 24 capas - **Parámetros**: ~335M parámetros - **Preentrenamiento**: Corpus multilingüe con énfasis en pares texto-texto

#### 3.3.3.2 Rendimiento en Benchmarks MTEB

E5-Large ha demostrado rendimiento superior en el benchmark MTEB (Massive Text Embedding Benchmark), particularmente en tareas de recuperación semántica y clasificación de similaridad textual (Muennighoff et al., 2023). Su arquitectura optimizada para recuperación lo posiciona como una alternativa competitiva a modelos propietarios en aplicaciones especializadas.

Las evaluaciones internas del proyecto revelaron que E5-Large presentó desafíos significativos en el dominio específico de Microsoft Azure, mostrando métricas de recuperación de 0.0 en todas las categorías (Precision@5, Recall@5, NDCG@5), lo que indica dificultades para recuperar documentos relevantes en los primeros 10 resultados. Sin embargo, mostró el mejor rendimiento en métricas de generación RAG como faithfulness (0.5909), sugiriendo que sus representaciones vectoriales, aunque problemáticas para recuperación, mantienen calidad semántica para tareas de generación de respuestas.

## 3.4 Arquitecturas RAG (Retrieval-Augmented Generation)

### 3.4.1 Fundamentos Teóricos de RAG

Las arquitecturas RAG combinan los beneficios de modelos parametrizados (conocimiento almacenado en parámetros) con acceso dinámico a conocimiento no parametrizado (bases de datos externas). Esta hibridación permite superar limitaciones fundamentales de los modelos de lenguaje tradicionales, incluyendo obsolescencia de información, alucinaciones factuales, y limitaciones de memoria (Lewis et al., 2020).

El paradigma RAG descompone la generación de respuestas en dos componentes diferenciables: 1. **Retriever**: Módulo especializado en recuperación de información relevante 2. **Generator**: Modelo de lenguaje que sintetiza respuestas utilizando información recuperada

### 3.4.2 Taxonomía de Arquitecturas RAG

#### 3.4.2.1 RAG Clásico (Retrieval-then-Generate)

La arquitectura RAG clásica implementa un pipeline secuencial donde la recuperación precede completamente a la generación. Esta aproximación es computacionalmente eficiente y facilita la interpretabilidad al separar claramente las responsabilidades de cada componente.

# Pipeline RAG implementado en el proyecto  
def generate\_rag\_answer(question: str, context\_docs: list):  
 # Preparar contexto desde documentos recuperados   
 context\_text = "\n\n".join([  
 f"Document {i+1}: {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:  
 """  
   
 response = client.chat.completions.create(  
 model="gpt-3.5-turbo",  
 messages=[{"role": "user", "content": prompt}],  
 max\_tokens=200,  
 temperature=0.1  
 )  
   
 return response.choices[0].message.content.strip()

#### 3.4.2.2 RAG Iterativo y Adaptativo

Variantes más sofisticadas incluyen RAG iterativo, donde el proceso de recuperación puede repetirse basándose en la generación parcial, y RAG adaptativo, donde el modelo aprende dinámicamente cuándo y cómo utilizar información externa (Jiang et al., 2023).

### 3.4.3 Métricas de Evaluación RAG

La evaluación de sistemas RAG requiere métricas especializadas que capturen tanto la calidad de recuperación como la calidad de generación. El framework RAGAS (Retrieval Augmented Generation Assessment) proporciona métricas comprehensivas incluyendo:

* **Faithfulness**: Consistencia factual entre respuesta generada y contexto
* **Answer Relevancy**: Relevancia de la respuesta respecto a la pregunta
* **Context Precision**: Precisión del contexto recuperado
* **Context Recall**: Completitud del contexto respecto al ground truth

## 3.5 CrossEncoders y Reranking

### 3.5.1 Fundamentos Teóricos del Reranking Neural

El reranking neural utiliza modelos que procesan conjuntamente consulta y documento, capturando interacciones semánticas más sofisticadas que los enfoques de embedding independientes. Los CrossEncoders representan el estado del arte en esta aproximación, utilizando mecanismos de atención cruzada para modelar relaciones complejas entre consulta y documento (Nogueira & Cho, 2019).

### 3.5.2 Arquitectura CrossEncoder ms-marco-MiniLM-L-6-v2

#### 3.5.2.1 Especialización en MS MARCO

El modelo ms-marco-MiniLM-L-6-v2 ha sido específicamente fine-tuneado en el dataset MS MARCO, que contiene 8.8 millones de pares pregunta-pasaje derivados de consultas reales de Bing. Esta especialización resulta en un modelo optimizado para escenarios de recuperación de información factual y técnica.

**Características técnicas verificadas:** - **Arquitectura base**: MiniLM-L6 (6 capas transformer) - **Tamaño**: ~90MB - **Longitud máxima**: 512 tokens por entrada - **Salida**: Score de relevancia (logit)

#### 3.5.2.2 Implementación de Normalización Min-Max

La normalización Min-Max aplicada a los scores del CrossEncoder garantiza comparabilidad entre consultas y estabilidad en las métricas de evaluación:

def rerank\_with\_cross\_encoder(question: str, documents: list, cross\_encoder, top\_k: int = 10):  
 # Preparar pares query-documento  
 pairs = []  
 for doc in documents:  
 content = doc.get('content', '')  
 pairs.append([question, content])  
   
 # Obtener scores del CrossEncoder  
 scores = cross\_encoder.predict(pairs)  
   
 # Aplicar normalización Min-Max para convertir a rango [0,1]  
 scores = np.array(scores)  
 if len(scores) > 1 and scores.max() != scores.min():  
 normalized\_scores = (scores - scores.min()) / (scores.max() - scores.min())  
 else:  
 normalized\_scores = np.full\_like(scores, 0.5)  
   
 # Agregar scores a documentos y ordenar  
 for i, doc in enumerate(documents):  
 doc['crossencoder\_score'] = float(normalized\_scores[i])  
   
 return sorted(documents, key=lambda x: x['crossencoder\_score'], reverse=True)[:top\_k]

### 3.5.3 Teoría de Optimización Multi-Etapa

La combinación de recuperación densa con reranking neural implementa una estrategia de optimización multi-etapa que balancea eficiencia computacional con precisión. La primera etapa (dense retrieval) opera como un filtro eficiente sobre el corpus completo, mientras la segunda etapa (reranking) aplica un modelo más sofisticado sobre un conjunto reducido de candidatos.

Esta aproximación es teóricamente sólida desde la perspectiva de optimización computacional, ya que permite aplicar modelos costosos únicamente sobre subconjuntos relevantes identificados por heurísticas eficientes (Chen et al., 2022).

## 3.6 Bases de Datos Vectoriales

### 3.6.1 Fundamentos de Búsqueda de Vectores de Alta Dimensión

La búsqueda eficiente en espacios vectoriales de alta dimensión presenta desafíos computacionales únicos relacionados con la maldición de la dimensionalidad y la necesidad de índices especializados. Los algoritmos de búsqueda exacta como fuerza bruta escalan linealmente con el tamaño del corpus, resultando impracticables para aplicaciones de producción.

### 3.6.2 Algoritmos de Búsqueda Aproximada

#### 3.6.2.1 Hierarchical Navigable Small World (HNSW)

HNSW implementa una estructura de grafo multicapa que permite búsqueda logarítmica aproximada en espacios de alta dimensión (Malkov & Yashunin, 2018). El algoritmo construye una jerarquía de grafos donde cada nivel contiene una fracción de los nodos del nivel inferior, permitiendo navegación eficiente desde búsqueda gruesa a refinada.

La estructura HNSW ofrece garantías teóricas de complejidad O(log N) para búsqueda y O(N log N) para construcción del índice, donde N es el número de vectores almacenados.

#### 3.6.2.2 Optimizaciones para Dominios Técnicos

En dominios técnicos especializados como documentación de Microsoft Azure, la distribución de vectores puede presentar características que permiten optimizaciones específicas. La clustering temática natural de documentos relacionados puede explotarse mediante técnicas de particionamiento inteligente del espacio vectorial.

### 3.6.3 ChromaDB: Arquitectura y Optimizaciones

#### 3.6.3.1 Decisión Arquitectónica: ChromaDB vs Weaviate

La migración de Weaviate a ChromaDB se fundamentó en criterios de optimización para flujos de investigación y desarrollo:

**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 (formato Parquet), simplicidad de configuración - **Aplicabilidad**: Óptimo para investigación y desarrollo iterativo

# Configuración ChromaDB implementada  
class EmbeddedRetriever:  
 def search(self, query\_embedding: np.ndarray, top\_k: int = 10):  
 # Calcular similitudes coseno  
 similarities = cosine\_similarity(query\_embedding.reshape(1, -1), self.embeddings)[0]  
   
 # Obtener top-k índices  
 top\_indices = np.argsort(similarities)[::-1][:top\_k]  
   
 results = []  
 for idx in top\_indices:  
 if idx < len(self.df):  
 doc = self.df.iloc[idx]  
 results.append({  
 'rank': len(results) + 1,  
 'cosine\_similarity': float(similarities[idx]),  
 'link': doc.get('link', ''),  
 'title': doc.get('title', ''),  
 'content': doc.get('content', '')  
 })  
   
 return results

#### 3.6.3.2 Arquitectura de Almacenamiento Multi-Modelo

El sistema implementa una arquitectura de almacenamiento que mantiene colecciones separadas para cada modelo de embedding, permitiendo comparaciones directas mientras preserva optimizaciones específicas por modelo:

* **Colecciones de documentos**: docs\_ada, docs\_mpnet, docs\_minilm, docs\_e5large
* **Colecciones de preguntas**: questions\_ada, questions\_mpnet, questions\_minilm, questions\_e5large
* **Ground truth**: questions\_withlinks (2,067 pares validados)

Esta arquitectura facilita evaluaciones comparativas rigurosas y permite optimizaciones independientes por modelo sin interferencia cruzada.

### 3.6.4 Consideraciones de Escalabilidad y Rendimiento

La selección de base de datos vectorial debe considerar múltiples factores incluyendo latencia de consulta, throughput, consumo de memoria, y capacidades de actualización incremental. Para corpus de tamaño moderado (~200K vectores), soluciones embebidas como ChromaDB ofrecen ventajas significativas en simplicidad operacional y rendimiento de consulta.

Para aplicaciones de producción con corpus de mayor escala (>1M vectores), bases de datos distribuidas como Weaviate, Pinecone, o Milvus se vuelven necesarias para mantener latencias aceptables y capacidades de escalamiento horizontal.

El proyecto demostró características de rendimiento sustanciales a través de múltiples dimensiones. El procesamiento de evaluación total alcanzó 774.78 segundos (12.9 minutos) para evaluación multi-modelo comprehensiva de 4 modelos de embedding contra 187,031 documentos. El throughput de procesamiento de documentos alcanzó aproximadamente 241 documentos por segundo para generación de embeddings, mientras el procesamiento de consultas mantuvo 0.057 consultas por segundo durante evaluación. Los requerimientos de almacenamiento variaron significativamente según dimensionalidad del modelo, desde 1.05 GB para MiniLM (384 dimensiones) hasta 2.23 GB para Ada (1,536 dimensiones), totalizando 6.48 GB para todos los modelos. La aceleración GPU proporcionó mejoras de rendimiento de 10-50x cuando estuvo disponible, con utilización exitosa de hardware Google Colab T4. El sistema demostró escalabilidad manejando operaciones vectoriales de gran escala sobre colecciones de 187K+ documentos manteniendo rendimiento consistente de recuperación mediante operaciones ChromaDB optimizadas y formatos de almacenamiento basados en parquet eficientes.

## 3.7 Conclusiones del Marco Teórico

El marco teórico presentado establece los fundamentos científicos y técnicos que sustentan la arquitectura RAG desarrollada en este proyecto. La convergencia de modelos de embeddings especializados, arquitecturas de reranking neural, y bases de datos vectoriales optimizadas habilita sistemas de recuperación semántica que superan significativamente las capacidades de enfoques tradicionales basados en coincidencia léxica.

La selección específica de componentes (Ada, MPNet, MiniLM, E5-Large para embeddings; ms-marco-MiniLM-L-6-v2 para reranking; ChromaDB para almacenamiento vectorial) se fundamenta en criterios teóricos sólidos relacionados con optimización de rendimiento, eficiencia computacional, y especialización de dominio. Esta arquitectura proporciona una base robusta para evaluación empírica de diferentes aproximaciones a la recuperación de información técnica especializada.

Los principios teóricos establecidos en este capítulo guían tanto la implementación técnica como la metodología de evaluación presentadas en capítulos posteriores, asegurando que el desarrollo del sistema se fundamenta en conocimiento científico validado y mejores prácticas de la industria.