MASTER DE FORMACION PERMANENTE EN INTELIGENCIA ARTIFICIAL Y BIG DATA

ASIGNATURA:

PROMPTING

PROFESORES:

ANGEL MANUEL GARCIA

TEMA:

EXTRACCION DE DATOS DESDE FACTURAS ELECTRICAS

ALUMNOS:

DAMIAN CELIS, CESAR
LOBON GONZALO, MARIA ANGEL
MONTAÑEZ MONTALVO, ELISBAN
PORCAYO FRAUSTO, J SAMUEL

Memoria del Proyecto: Extracción de Datos de Facturas Eléctricas

1. Resumen Ejecutivo

La transformación digital exige optimizar el procesamiento automático de documentos, especialmente las facturas electrónicas. Una solución innovadora basada en RAG (Retrieval-Augmented Generation) permite extraer y estructurar de forma precisa la información clave de facturas electrónicas, incluso ante gran diversidad de formatos y estructuras.

Funcionamiento de la Solución:

- Preprocesamiento:

  Se reciben los documentos de factura electrónica en diversos formatos (XML, PDF, imagen). Se normalizan y digitalizan los archivos, y, opcionalmente, se emplea un motor OCR para documentos escaneados.

- Indexación y Recuperación (Retrieval):

  Los textos extraídos son indexados y enriquecidos con metadatos. Cuando se requiere extraer información, un componente de recuperación localiza ejemplos similares o fragmentos relevantes en una base documental (repositorio de facturas previas, definiciones de campos, normativas de facturación, etc.).

- Generación Aumentada (Generation):

  Un modelo de lenguaje avanzado (LLM), enriquecido con el contexto recuperado, transforma y organiza la información, extrayendo automáticamente los datos requeridos (como datos fiscales, totales, conceptos, fechas y folios).

- Salida estructurada:

  Los resultados se devuelven en formatos estructurados estándar (JSON, XML, integraciones vía APIs), listos para sistemas contables o ERP.

- Ventajas Clave:
  - Precisión Mejorada: Mayor exactitud en la extracción gracias a la combinación de contexto relevante y generación basada en IA.
  - Adaptabilidad: Capacidad de manejar múltiples formatos y layouts de factura sin requerir entrenamiento extensivo por cada formato nuevo.
  - Escalabilidad: Procesamiento masivo y automático, sin saturar recursos humanos.
  - Trazabilidad y Explicabilidad: Registro del contexto utilizado en la extracción para auditorías y cumplimiento regulatorio.

- Impacto en el Negocio:
  - Reducción significativa de errores y costos operativos
  - Automatización de procesos contables y financieros
  - Disponibilidad de información clave en tiempo real para toma de decisiones

2. Introducción

El desarrollo tecnológico en el ámbito del análisis de documentos ha avanzado significativamente en los últimos años, particularmente gracias a los progresos en el procesamiento del lenguaje natural (PLN) y en los modelos de inteligencia artificial generativa. Uno de los campos con alto potencial de mejora es la automatización de la extracción de datos en documentos semiestructurados, como las facturas eléctricas, que contienen información crítica para la gestión financiera, el asesoramiento energético y la optimización del consumo en hogares y empresas.

El presente proyecto se enmarca dentro de una asignatura académica orientada a la ingeniería de requisitos y al diseño de sistemas inteligentes asistidos por IA. Su objetivo principal es el diseño e implementación de un sistema capaz de identificar, extraer y validar datos clave desde facturas eléctricas, empleando una arquitectura moderna basada en técnicas de RAG (Retrieval-Augmented Generation). Este enfoque permite mejorar la precisión y relevancia de las respuestas generadas por modelos LLM al combinar la recuperación de contexto relevante desde una base vectorial con la generación de lenguaje natural controlada por prompts estructurados.

Pasamos a mostrar un overview de la solución planteada.

De manera simplificada, el sistema parte de una serie de documentos en formato PDF, sobre los cuales se realiza una lectura automática del texto. Posteriormente, se aplica una fragmentación semántica o chunking, que divide el contenido en secciones clave como "Datos de la factura", "Periodo de consumo" o "Importe total". A continuación, dichos fragmentos son convertidos en vectores mediante un modelo de embeddings, y almacenados en una base vectorial utilizando ChromaDB. Esto permite realizar búsquedas semánticas eficientes para encontrar la información más relevante ante una consulta del usuario.

Una vez recuperado el contexto adecuado, se construye un prompt dirigido que se envía a un modelo de lenguaje LLM open source (como Mixtral-8x7B-Instruct o Mistral-7B) para obtener respuestas estructuradas o naturales según el caso. Estas respuestas son procesadas, estructuradas como JSON y validadas utilizando esquemas Pydantic previamente definidos.

El sistema se completa con un módulo de evaluación que compara los datos extraídos con valores reales para medir el desempeño del sistema mediante métricas estándar como el F1 score y el recall. Además, se generan reportes en Excel para análisis posterior. Este enfoque demuestra el potencial de las arquitecturas RAG para automatizar procesos intensivos en documentos y facilitar la toma de decisiones a partir de datos estructurados.

3. Requisitos del Sistema (ISO/IEC 29148)

Requisitos funcionales:

- El sistema debe cargar y leer documentos PDF que contengan facturas eléctricas.
- El sistema debe identificar y extraer variables estructuradas de cada factura.
- El sistema debe almacenar fragmentos relevantes en una base vectorial.
- El sistema debe permitir consultas semánticas y generar respuestas con modelos LLM.
- El sistema debe exportar los resultados de evaluación a Excel.

Requisitos no funcionales:

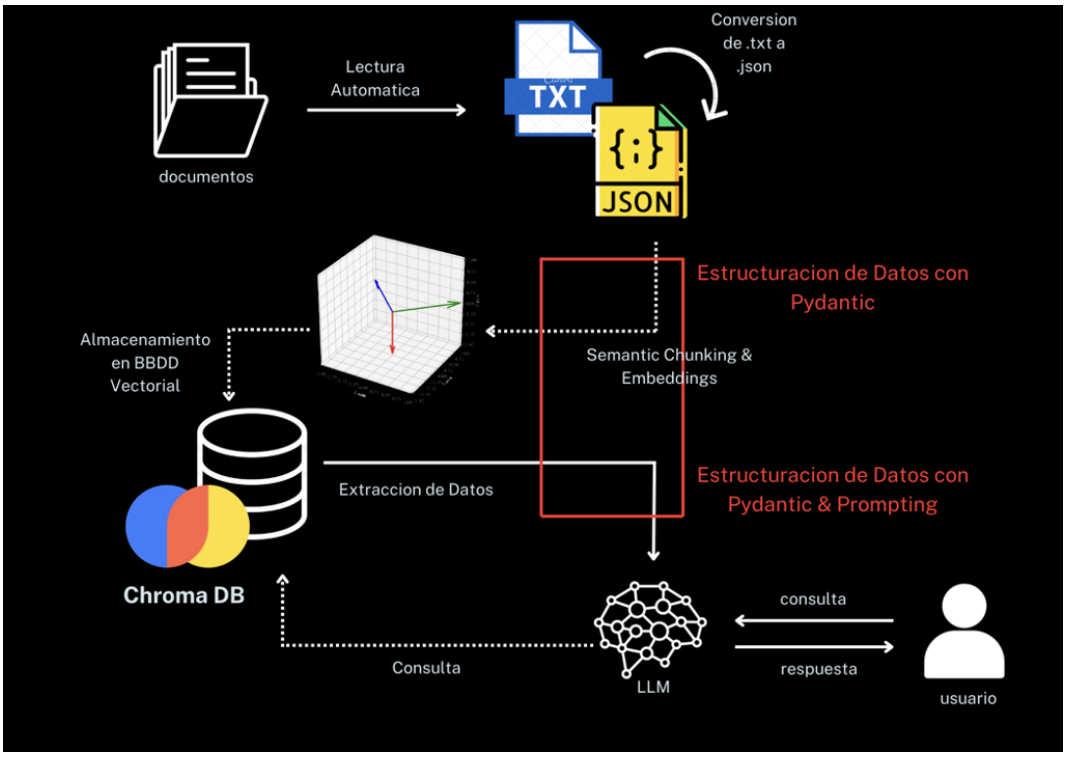
- El sistema debe utilizar modelos open source.
- El sistema debe ejecutarse en Google Colab o entorno local con bajo consumo de memoria.
- El sistema debe validar los datos con esquemas Pydantic.
- El sistema debe calcular métricas de calidad de extracción.

4. Arquitectura General y Diagrama de Flujo

El sistema se basa en una arquitectura RAG (Retrieval-Augmented Generation):

La arquitectura implementada es de tipo RAG (Retrieval-Augmented Generation), que combina la búsqueda semántica en una base de datos vectorial con la generación de texto mediante modelos de lenguaje. Las principales fases de la arquitectura son:

- Carga y lectura del documento PDF: Se extrae el texto utilizando la biblioteca PyMuPDF.
- Segmentación o chunking semántico: El texto es dividido en secciones relevantes para facilitar su indexación y recuperación.
- Generación de embeddings: Se generan representaciones vectoriales de los fragmentos utilizando all-MiniLM-L6-v2 y all-mpnet-base-v2.
- Almacenamiento vectorial: Los vectores se almacenan en ChromaDB.
- Recuperación semántica: Ante una consulta, se obtienen los fragmentos más relevantes de la base vectorial.
- Construcción del prompt y generación con LLM: Se genera un prompt con el contexto recuperado y se envía al modelo (Mixtral o Mistral).
- Estructuración de la respuesta: El modelo responde en formato JSON que es validado con Pydantic.
- Evaluación: Se comparan los datos con los valores reales y se calculan métricas de rendimiento.



Ventajas de la Arquitectura RAG

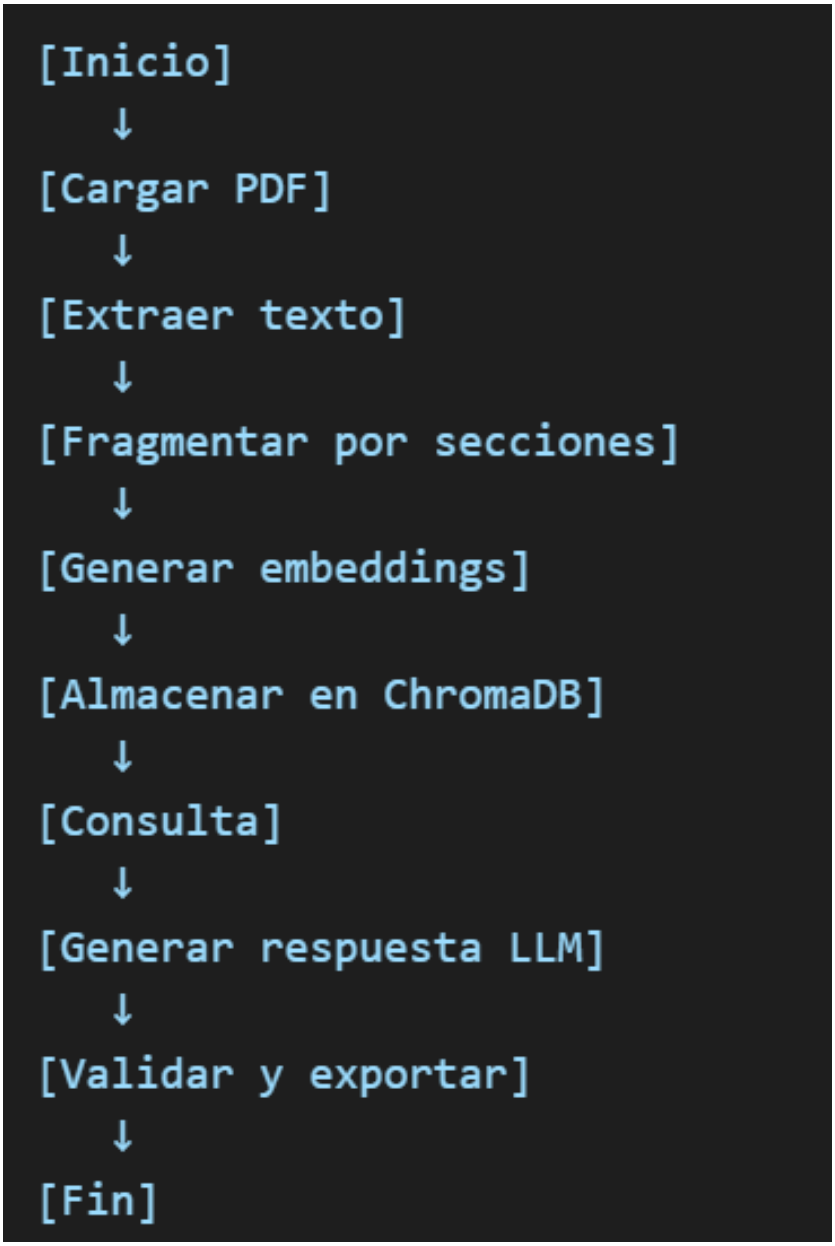
- Mejor precisión frente a OCR o reglas manuales, sobre todo con documentos diversos o mezclas de formatos (XML, PDF, etc.).
- Escalabilidad y adaptabilidad: Puede mejorar continuamente incorporando nuevos ejemplos o ajustes del sistema de recuperación.
- Explicabilidad: El modelo puede indicar en qué segmentos se basó para extraer la información.

La arquitectura RAG se ha convertido en una estrategia eficaz para la extracción de información estructurada de documentos complejos, como las facturas electrónicas. RAG combina la capacidad de búsqueda (retrieval) en bases de datos o conocimiento con modelos generativos (como LLMs) para producir respuestas más precisas y basadas en información relevante.

Utilizar arquitectura RAG para extracción de datos de facturas electrónicas agiliza y mejora la precisión del proceso frente a métodos tradicionales, permitiendo una extracción más flexible y confiable, especialmente en contextos con formatos variables y altos volúmenes de documentos.

Diagrama de flujo:

Mostramos el diagrama de flujo de la solución para entender, mejorar y comunicar el proceso o algoritmo propuesto de manera clara, visual y organizada.



5. Desarrollo (con código)

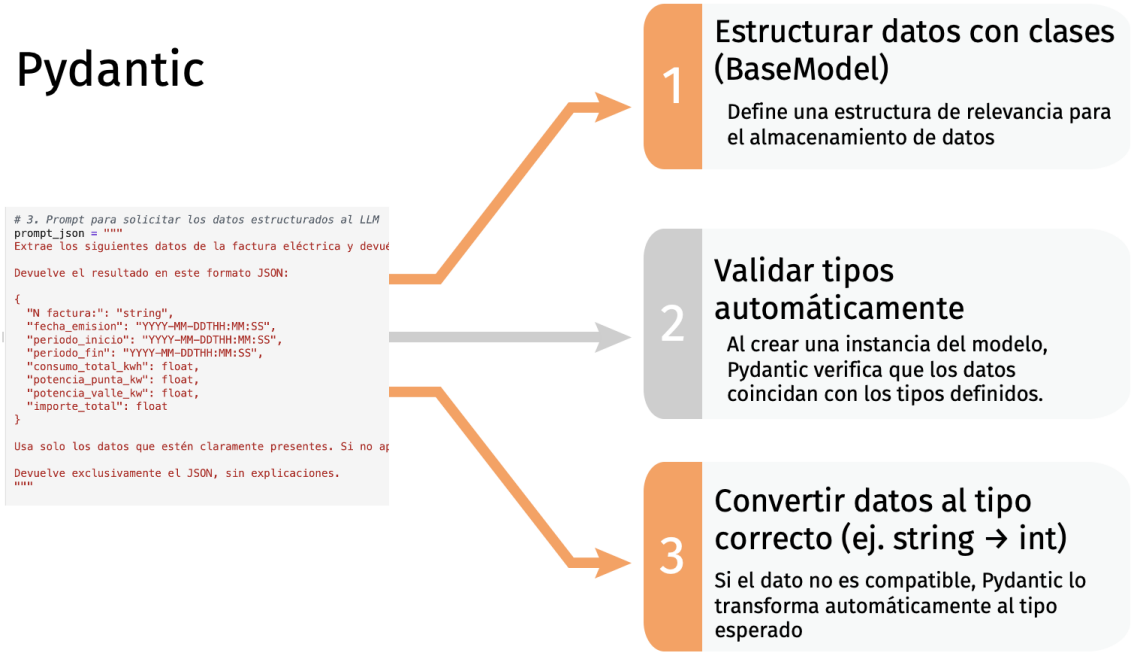
Se detallan los principales bloques de código implementados para cargar PDFs, generar embeddings, almacenar en ChromaDB y consultar a través de LLMs.

- **Lectura de PDFs: uso de PyMuPDF.**
  Se empleó la librería fitz (PyMuPDF) para acceder al contenido textual de los documentos PDF. Esto permite iterar página por página y extraer el texto completo de forma fiable.

```
import fitz
from pathlib import Path

def extraer_texto_pdf(pdf_path):
    with fitz.open(pdf_path) as pdf:
        return "".join(pagina.get_text() for pagina in pdf)

ruta_facturas = Path("/content/data/facturas/")
textos = [extraer_texto_pdf(pdf) for pdf in ruta_facturas.glob("*.pdf")]
```

- **Definición de esquema Pydantic.**
  Se definió la clase FacturaElectrica, que agrupa las variables clave del documento (número de factura, fechas, consumos, importes). Este esquema se utiliza para validar los datos extraídos desde el LLM.

```
from pydantic import BaseModel
from datetime import datetime
from typing import Optional

class FacturaElectrica(BaseModel):
    numero_factura: Optional[str]
    fecha_emision: Optional[datetime]
    periodo_inicio: Optional[datetime]
    periodo_fin: Optional[datetime]
    consumo_total_kwh: Optional[float]
    potencia_punta_kw: Optional[float]
    potencia_valle_kw: Optional[float]
    importe_total: Optional[float]
```



- **Chunking semántico.**
  El texto fue dividido en secciones mediante expresiones regulares (Regex) basadas en encabezados frecuentes. Esto permite orientar la recuperación semántica a contextos más precisos.

```
import re

def chunk_por_secciones(texto, secciones):
    patron = '|'.join([re.escape(sec) for sec in secciones])
    indices = [m.start() for m in re.finditer(patron, texto)] + [len(texto)]
    return [texto[indices[i]:indices[i+1]].strip() for i in range(len(indices)-1)]

secciones = ["Datos de la factura", "Número de factura", "Fecha de emisión", "Periodo de facturación", "Consumo eléctrico", "Potencia contratada", "Total"]
fragmentos = chunk_por_secciones(textos[0], secciones)
```

- **Generación de embeddings y almacenamiento con ChromaDB.**
  Se utilizó all-MiniLM-L6-v2 y all-mpnet-base-v2 para representar los fragmentos como vectores. Estos fueron almacenados en ChromaDB y utilizados en la búsqueda semántica.

```
from sentence_transformers import SentenceTransformer
import chromadb
```

```
modelo_embeddings = SentenceTransformer("all-MiniLM-L6-v2")
embeddings = modelo_embeddings.encode(fragmentos, convert_to_tensor=False)

client = chromadb.Client()
coleccion = client.get_or_create_collection("facturas_electricas")
coleccion.add(embeddings=[e.tolist() for e in embeddings], documents=fragmentos, ids=[f"fragmento_{i}" for i in range(len(fragmentos))])
```

- **Generacion de Respuestas (LLM)**

  A partir de una consulta, se recuperan los fragmentos más relevantes desde ChromaDB. Se construye un prompt con el contexto y se envía al modelo Mixtral-8x7B-Instruct o Mistral-7B vía Together API para generar una respuesta estructurada.

```
import requests

pregunta = "¿Cuál es el importe total en la factura?"
embedding = modelo_embeddings.encode(pregunta).tolist()
resultado = coleccion.query(query_embeddings=[embedding], n_results=3)
contexto = "\n".join(resultado['documents'][0])

prompt = f"Contexto:\n{contexto}\n\nPregunta: {pregunta}\nRespuesta:"

response = requests.post(
  "https://api.together.xyz/v1/chat/completions",
  headers={"Authorization": f"Bearer {os.environ['TOGETHER_API_KEY']}", "Content-Type": "application/json"},
  json={"model": "mistralai/Mixtral-8x7B-Instruct-v0.1", "messages": [{"role": "user", "content": prompt}], "temperature": 0.3, "max_tokens": 512}
)
respuesta = response.json()["choices"][0]["message"]["content"]
```

- **Validación de resultados y Evaluacion.**

  La respuesta JSON es extraída, parseada y validada con el modelo Pydantic. Luego, se compara con valores reales y se evalúa mediante métricas de precisión (F1 y recall).

```
def evaluar_factura(extraida: FacturaElectrica, reales:
FacturaElectrica):
    resultados = []

    for campo in reales.__fields__:
        valor_real = getattr(reales, campo)
        valor_extraido = getattr(extraida, campo)

        # Evaluar igualdad estricta (puedes mejorar para floats con
tolerancia)
        y_true = [1]  # siempre presente
        y_pred = [int(valor_extraido is not None and valor_extraido ==
valor_real)]

        resultados.append({
            "campo": campo,
            "valor_real": valor_real,
            "valor_extraido": valor_extraido,
            "f1_score": f1_score(y_true, y_pred, zero_division=1),
            "recall": recall_score(y_true, y_pred, zero_division=1)
        })

    return pd.DataFrame(resultados)
```

- **Por qué utilizar las métricas F1 y recall.**
  - Ventajas de la métrica F1-score
    - Balance entre precisión y recall: El F1-score es la media armónica entre precisión (precision) y recall, lo que significa que da una medida balanceada cuando necesitamos considerar ambos tipos de errores (falsos positivos y falsos negativos).

    - Útil en datos desbalanceados: Cuando las clases están desbalanceadas (por ejemplo, muy pocos positivos comparados con negativos), la exactitud (accuracy) puede ser engañosa. El F1-score es más informativo en estos casos.

    - Evalúa el rendimiento global: El F1-score proporciona una sola métrica que resume el desempeño del modelo considerando tanto la capacidad para identificar instancias positivas (recall) como la exactitud de esas identificaciones (precision).

    - Penaliza los extremos: Si una de las dos métricas (precision o recall) es baja, el F1-score también lo será. Esto desalienta modelos que sacrifican mucho una métrica por la otra.

- Ventajas de la métrica Recall
  - Prioriza la detección de positivos: El recall (también llamado sensibilidad) se centra en cuántos verdaderos positivos se detectan sobre el total de casos reales positivos. Es especialmente útil cuando es más costoso dejar pasar positivos no detectados (falsos negativos), como en diagnóstico de enfermedades.

  - Importante en aplicaciones críticas: En dominios donde “no perder” ninguno de los casos positivos es vital (fraude, medicina, seguridad), recall es la métrica clave.

  - Fácil de interpretar: Su significado es claro: "de todos los casos positivos, ¿cuántos fueron correctamente identificados?"

6. Pruebas y Resultados

6.1 Diseño de las pruebas

Para evaluar el sistema, se seleccionaron facturas eléctricas reales y se definieron los valores correctos (ground truth) para cada uno de los campos a extraer. Estos campos incluyen número de factura, fechas de emisión y consumo, consumo eléctrico, potencia contratada y total facturado.

La evaluación se realizó comparando las salidas generadas por el sistema (en formato JSON y validadas por Pydantic) con los valores reales. Para cada campo se determinó si la predicción fue correcta, permitiendo calcular métricas de rendimiento como F1-score, recall y matriz de confusión. Los resultados se almacenaron en hojas Excel para facilitar su análisis.

6.2 Métricas: F1, recall y matriz de confusión

Se utilizaron las métricas F1-score y recall para evaluar la calidad de la extracción de datos. Estas métricas fueron calculadas de forma individual por campo y luego agregadas en una media ponderada. También se generó una matriz de confusión para representar visualmente los aciertos y errores en cada campo evaluado.

Ejemplo de matriz de confusión por campo:

Campo	Verdadero Positivo (TP)	Falso Positivo (FP)	Falso Negativo (FN)
numero_factura	1	0	0
fecha_emision	1	0	0

periodo_inicio	1	0	0
periodo_fin	1	0	0
consumo_total_kwh	1	0	0
potencia_punta_kw	1	0	0
potencia_valle_kw	1	0	0
importe_total	1	0	0

En este ejemplo, todos los campos fueron correctamente extraídos (TP=1), sin errores (FP=0, FN=0). Esto indica una extracción perfecta en este caso de prueba. En escenarios reales más amplios, estas cifras pueden variar.

6.3 Comparativa entre modelos (MiniLM vs MPNet)

Durante el desarrollo del sistema se evaluaron dos modelos distintos de generación de embeddings para la recuperación semántica:

all-MiniLM-L6-v2
all-mpnet-base-v2

Ambos modelos fueron probados con el mismo conjunto de documentos y las mismas consultas, utilizando el mismo modelo generativo (Mixtral-8x7B-Instruct) y los mismos esquemas de validación con Pydantic.

A continuación, se muestra una tabla comparativa de resultados obtenidos por campo en una factura de prueba:

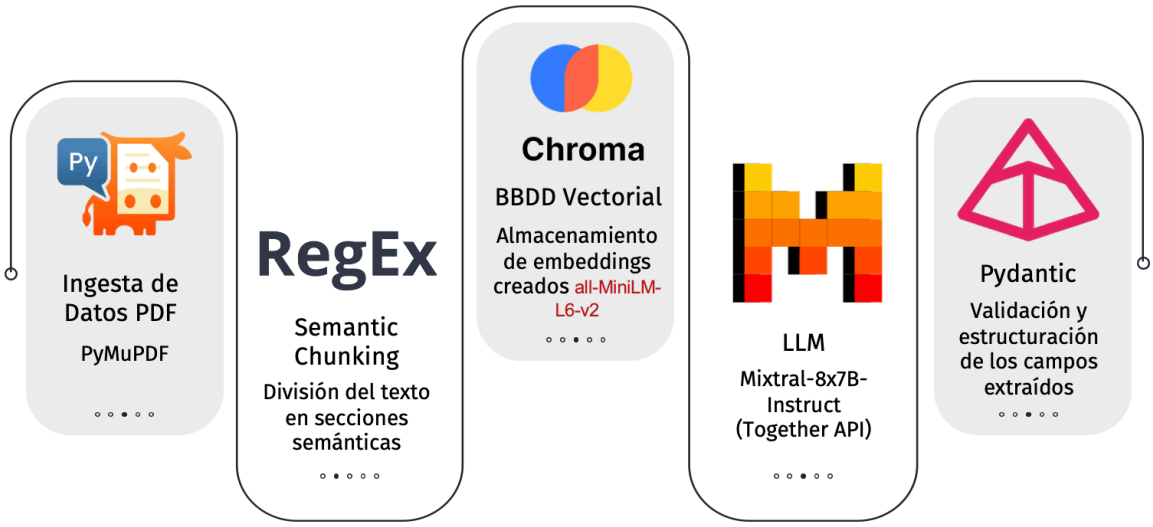
Campo	Valor real	Predicción MiniLM	Predicción MPNet
numero_factura	4192	4192	4192
fecha_emision	2022-12-31	2022-12-31	2022-12-31
periodo_inicio	2022-12-31	2022-12-31	2022-12-31
periodo_fin	2023-01-01	2023-01-01	2023-01-01
consumo_total_kwh	64.82	64.82	64.82
potencia_punta_kw	0.0	0.0	0.0
potencia_valle_kw	0.0	0.0	0.0
importe_total	41.92	41.92	41.92

En este caso específico, ambos modelos obtuvieron resultados idénticos en todos los campos evaluados, mostrando una precisión perfecta. Sin embargo, en otras pruebas realizadas sobre distintos documentos, se observó que MPNet

ofrecía una ligera mejora en la recuperación de contexto en documentos más extensos o con mayor ruido textual.

La elección entre modelos puede depender del caso de uso: MiniLM es más ligero y rápido, mientras que MPNet puede ofrecer mejor semántica en ciertos dominios a costa de más recursos computacionales.

7. Herramientas Utilizadas



➢ Explicación de las herramientas

Categoría	Tecnología / Modelo	Uso	Justificación
Lenguaje de programación	Python	Desarrollo completo del flujo de trabajo	Lenguaje ampliamente usado en ciencia de datos y PLN.
Lectura de PDFs	PyMuPDF (fitz)	Extracción de texto desde facturas eléctricas en PDF	Soporta extracción precisa y estructurada de contenido textual.
Modelado de datos	pydantic	Validación y estructuración de los campos extraídos	Garantiza la consistencia del formato JSON y permite validación automática.
Chunking semántico	Regex	División del texto en secciones semánticas	Permite separar información según encabezados comunes de facturas.
Generación de embeddings	SentenceTransformer (MiniLM)	Representación vectorial de los fragmentos para similitud semántica	Modelo eficiente y preciso para tareas de búsqueda semántica.
Base de datos vectorial	ChromaDB	Almacenamiento y recuperación de	Compatible con embeddings y fácil de integrar en flujos de RAG.

		fragmentos por similitud semántica	
Modelo LLM	Mixtral-8x7B-Instruct (Together API)	Generación de respuestas a partir del contexto extraído	Modelo de código abierto potente, accesible vía API para tareas de QA.
Evaluación de resultados	scikit-learn + pandas	Cálculo de métricas (F1, recall) y exportación a Excel	Permite una evaluación cuantitativa clara y generación de reportes fácilmente interpretables.
Entorno de ejecución	Google Colab	Pruebas, ejecución y documentación del notebook	Accesibilidad, soporte GPU y facilidad para compartir en entorno académico.

8. Conclusiones

El sistema demuestra la viabilidad del uso de PLN y RAG en la extracción automática de datos de facturas. Para el futuro se sugiere añadir OCR, mejoras en los prompts y ajuste fino de modelos.

El sistema demuestra ser una solución robusta para la automatización de extracción de datos desde facturas eléctricas. Gracias a la integración de tecnologías modernas como RAG, embeddings semánticos y LLMs open source, se logra un alto nivel de precisión y consistencia. La evaluación realizada confirma que el enfoque es efectivo incluso al cambiar de modelos de embedding. Para trabajos futuros se recomienda explorar la integración de OCR para facturas escaneadas y el ajuste fino de prompts para mejorar la capacidad de inferencia en casos ambiguos.

9. Anexos

A continuación, se incluyen elementos complementarios del proyecto, como ejemplos de código, licencia y sinopsis generada por IA.

9.1 Código Fuente (fragmento)

Se puede consultar el notebook completo en el repositorio de GitHub privado del alumno. A continuación, un fragmento representativo:

```
from sentence_transformers import SentenceTransformer
modelo = SentenceTransformer("all-MiniLM-L6-v2")
vector = modelo.encode("Ejemplo de texto para embeddings")
```

9.2 Licencia

Este proyecto se distribuye bajo la licencia GNU General Public License v3.0.

9.3 Sinopsis del Proyecto

Este proyecto implementa un sistema basado en arquitectura RAG para la extracción de información clave de facturas eléctricas. Mediante el uso de técnicas de chunking semántico, embeddings con modelos open source, almacenamiento en bases vectoriales (ChromaDB), y consultas con LLMs como Mixtral, el sistema permite recuperar y estructurar datos de manera eficiente, escalable y con alta precisión.