Prototipo INEGI-LLM con RAG

Este repositorio contiene un prototipo técnico que demuestra cómo un LLM con RAG puede **leer y estructurar** la información de las hojas de diseño de la ENA 2025 para generar metadatos JSON, clases Java y documentos de soporte. Además, permite **interactuar en modo chat** con esos metadatos usando un flujo básico de recuperación y generación. El proyecto es capaz de:

- Generar archivos JSON de metadatos por tabla.
- Generar clases Java de dominio a partir de esos metadatos.
- Construir documentos de texto con información de tablas y columnas.
- Consultar metadatos mediante un chat en consola respaldado por un LLM con RAG simplificado.

El documento PDF adjunto describe el diseño conceptual de fondo.
Este README se enfoca en explicar la estructura del proyecto y los pasos concretos para ejecutarlo.

---

1. Alcance del prototipo

El prototipo realiza las siguientes tareas:

- Leer una hoja de diseño en Excel con la estructura típica de la ENA (nombre de tabla, campo, tipo, longitud, precisión, función, nivel de desagregación, etc.).
- Construir un metadato estructurado por tabla y guardarlo en un archivo JSON.
- Validar de forma básica los metadatos generados.
- Generar una clase Java de dominio por tabla, con un campo por columna.
- Construir documentos de texto a partir de los metadatos (tabla y columnas) para usarlos como contexto en consultas.
- Permitir un chat en consola donde el usuario formula preguntas sobre tablas y columnas y recibe respuestas basadas en los metadatos.

Quedan pendientes para el futuro:

- Integración con sistemas de captura, carga, codificación, normalización o validación específicos ya existentes, esto se trabajara para cada proyecto, ahora se aborda un proceso en especifico.
- Esquemas de seguridad, autenticación o despliegue institucional.
- Indexadores vectoriales o infraestructura de RAG en producción.
- Entrenamiento real de modelos de lenguaje con datos masivos del INEGI.

El código se entrega como una base técnica que puede ser extendida por otros equipos.

---

2. Requisitos previos

2.1 Python

- Python 3.10 o superior.

- Entorno virtual recomendado.

Pasos sugeridos:

```
cd inegi-meta-llm
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

Dependencias principales:

- `pandas`, `openpyxl` para lectura de Excel.
- `pytest` para pruebas.
- `transformers` y `torch` para usar modelos de Hugging Face.

2.2 Java y Maven

- JDK 8 o superior (se ha probado con versiones modernas).
- Maven 3.9 o superior.

Verificación rápida:

```
mvn -v
java -version
```

En la carpeta `java` se incluye un `pom.xml` con la configuración necesaria para:

- Compilar las clases Java del prototipo.
- Ejecutar la clase `MainGenerateFromJson` mediante `mvn exec:java`.

Archivos de ejemplo no incluidos en el repositorio:

Por tratarse de información interna del INEGI, los archivos de diseño reales **no se incluyen** en este repositorio público.

El flujo descrito en este documento asume la existencia de los siguientes archivos, que cada usuario debe colocar manualmente en su entorno local:

- Carpeta `excel/`
  Aquí deben colocarse las hojas de diseño en formato Excel que se vayan a procesar.
  En los ejemplos se hace referencia a un archivo como
  `excel/UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx`,
  pero ese archivo **no viaja en el repositorio** y debe ser proporcionado por quien ejecute el prototipo.
- Carpeta `tests/data/` (opcional, solo para ejecutar `pytest`)
  Si se desean correr las pruebas unitarias, es necesario copiar manualmente:

  - Un Excel de diseño renombrado como `tests/data/diseno_tabla_ejemplo.xlsx`.
  - Un JSON de metadatos renombrado como `tests/data/metadatos_tr_ejemplo.json` (se puede generar con el propio flujo Excel -> JSON descrito en este README). En caso de que en el futuro el repositorio se migre a un entorno privado o controlado, estos archivos podrían compartirse de forma interna, pero mientras el repositorio sea público se opta por **no exponer directamente archivos de diseño ni metadatos reales**.

3. Estructura del proyecto

Vista general de carpetas y archivos principales:

- `excel/`
  - `UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx`
    Archivo de ejemplo de hoja de diseño ENA 2025.
- `output/`
  - `metadata_TR_ENA2025_CUEST_UP1.json`
    Metadatos JSON generados desde Excel para una tabla.
  - `docs_TR_ENA2025_CUEST_UP1.json`
    Documentos de tabla y columnas generados a partir del JSON de metadatos.
- `python/`
  - `metadata_model.py`
    Define las clases `TableMetadata` y `ColumnMetadata` en Python.
  - `metadata_validator.py`
    Aplica validaciones básicas sobre los metadatos generados.
  - `excel_to_metadata.py`
    Contiene la lógica para leer el Excel y construir un `TableMetadata`.
  - `cli_generate_metadata.py`
    Proporciona la interfaz de línea de comandos para pasar de Excel a JSON.
  - `metadata_to_rag_docs.py`
    Convierte el JSON de metadatos en documentos de texto para el RAG.
  - `__init__.py`
- `java/`
  - `pom.xml`
    Definición del proyecto Maven y dependencias (Jackson, plugin exec).
  - `src/main/java/mx/inegi/meta/model/ColumnMetadata.java`
    Modelo Java para columnas de metadatos.
  - `src/main/java/mx/inegi/meta/model/TableMetadata.java`
    Modelo Java para tablas de metadatos.
  - `src/main/java/mx/inegi/meta/io/JsonMetadataLoader.java`
    Utilidad para leer el JSON de metadatos y convertirlo en objetos Java.
  - `src/main/java/mx/inegi/meta/generator/JavaClassGenerator.java`
    Generador de clases Java a partir de un `TableMetadata`.

    - `src/main/java/mx/inegi/meta/MainGenerateFromJson.java`
      Punto de entrada para ejecutar la generación de clases con Maven.
    - `src/generated/mx/inegi/generated/`
      Carpeta donde se escriben las clases Java generadas, por ejemplo:
      - *`TR_ENA2025_CUEST_UP1.java`*

- `llm_stub/`

    - `metadata_document_builder.py`
      Construye una lista de documentos RAG a partir de un JSON de metadatos.
    - `simple_retriever.py`
      Permite buscar documentos por tabla, columna o palabra clave.
    - `local_llm_client.py`
      Cliente para el modelo de lenguaje (modo simulado o Hugging Face).
    - `chat_demo.py`
      Script de chat en consola para hacer preguntas sobre los metadatos.

- `tests/`

    - `test_excel_to_metadata.py`
      Verifica que desde un Excel se construye un `TableMetadata` válido.
    - `test_metadata_to_rag_docs.py`
      Verifica la generación de documentos de tabla y columnas desde el JSON.
    - `test_simple_retriever.py`
      Verifica que la búsqueda por tabla, columna y palabra clave funciona.

- `README.md`

4. Paso a paso – Excel a JSON de metadatos

4.1 Insumos en Excel

Se espera que las hojas de diseño en Excel contengan, para cada columna, campos como:

- Nombre de la tabla.
- Nombre de la columna.
- Tipo de dato.
- Longitud.
- Precisión.
- Función.
- Nivel de desagregación.

Colocar los archivos en la carpeta:

```
excel/ UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx
```

4.2 Generar JSON desde la línea de comandos

Desde la raíz del proyecto:

```
venv\Scripts\activate python -m python.cli_generate_metadata
```

El programa ejecuta lo siguiente:

1. Busca archivos `.xlsx` en la carpeta `excel` y muestra una lista para seleccionar uno.
2. Usa por defecto la primera hoja del archivo (por ejemplo "BLOQUE 1").
3. Identifica las tablas disponibles y permite elegir una.

Ejemplo de salida:

```
Se encontraron 5 tablas en la hoja 'BLOQUE 1'. Selecciona una:
  1. TRS_ENA2025_CONTROL_CUPA
  2. TR_ENA2025_CUEST_UP1
  3. TR_ENA2025_CUEST_UP2
  4. TR_ENA2025_CUEST_UP3
  5. VARIABLE GENERADA

Ingresa el número de la tabla a procesar: 2

Procesando tabla 'TR_ENA2025_CUEST_UP1' desde
UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx...
  Hoja: BLOQUE 1
  Columnas encontradas: 44
  Metadatos guardados en: output\metadata_TR_ENA2025_CUEST_UP1.json

Validando metadatos generados...
✓ Validación exitosa

Archivo generado correctamente: output\metadata_TR_ENA2025_CUEST_UP1.json
```

```
D:\jorge.[redacted]llalobos\Desktop\SCRIPTS\5 - inegi-meta-llm>python -m python.cli_generate_metadata
Se encontraron varios archivos Excel. Selecciona uno:
  1. UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx
  2. ~$UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx

Ingresa el número del archivo a usar: 1

Se encontraron 18 tablas en la hoja 'COMPLETO'. Selecciona una:
  1. TRS_ENA2025_CONTROL_CUPA
  2. TR_ENA2025_ABONO
  3. TR_ENA2025_APROTEGIDA
  4. TR_ENA2025_CABIERTO
  5. TR_ENA2025_CUEST_UP1
  6. TR_ENA2025_CUEST_UP2
  7. TR_ENA2025_CUEST_UP3
  8. TR_ENA2025_CUEST_UP4
  9. TR_ENA2025_CUEST_UP5
  10. TR_ENA2025_FERTILIZANTE
  11. TR_ENA2025_NUTRICION
  12. TR_ENA2025_UNIDADES_PRODUCCION
  13. TR_ENA2025_VAVES
  14. TR_ENA2025_VBOVINO
  15. TR_ENA2025_VCAPRINO
  16. TR_ENA2025_VOVINO
  17. TR_ENA2025_VPORCINO
  18. VARIABLE GENERADA

Ingresa el número de la tabla a procesar: 3

Procesando tabla 'TR_ENA2025_APROTEGIDA' desde UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx...
  Hoja: COMPLETO
  Columnas encontradas: 17
  Metadatos guardados en: output\metadata_TR_ENA2025_APROTEGIDA.json

Validando metadatos generados...
✓ Validación exitosa

Archivo generado correctamente: output\metadata_TR_ENA2025_APROTEGIDA.json
```

Resultado esperado:

- Un archivo JSON en `output/metadata_<NOMBRE_TABLA>.json`
  por ejemplo: `output/metadata_TR_ENA2025_CUEST_UP1.json`.

5. Paso a paso – JSON a clase Java

El siguiente paso convierte los metadatos JSON de una tabla en una clase Java generada.

5.1 Compilar el módulo Java

Ir a la carpeta `java`:

```
cd java
mvn compile
```

Maven descargará las dependencias de Jackson y compilará las clases en `target/classes`.

5.2 Generar la clase Java desde un JSON

Una vez compilado, ejecutar:

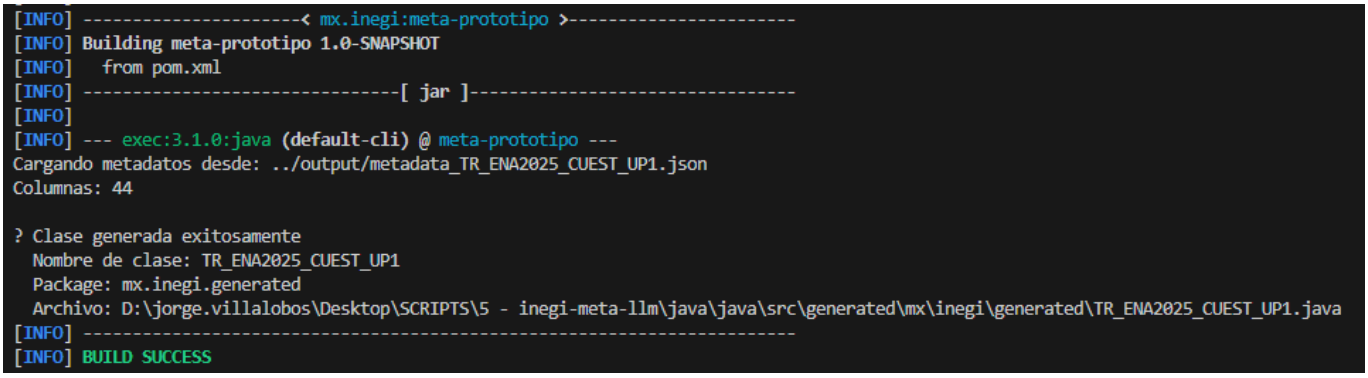
```
mvn exec:java -Dexec.args="../output/metadata_TR_ENA2025_CUEST_UP1.json
mx.inegi.generated"
```

Donde:

- El primer argumento es la ruta al JSON de metadatos (vista desde la carpeta `java`).
- El segundo argumento es el paquete donde se generará la clase (por ejemplo `mx.inegi.generated`).

Ejemplo de salida esperada:

```
Leyendo metadatos desde ../output/metadata_TR_ENA2025_CUEST_UP1.json
Tabla: TR_ENA2025_CUEST_UP1
Columnas: 44
Clase generada: TR_ENA2025_CUEST_UP1
Archivo generado en: src/generated/mx/inegi/generated/TR_ENA2025_CUEST_UP1.java
```



La clase generada debe contener:

- Declaración de paquete (por ejemplo `package mx.inegi.generated;`).
- Campos privados con los nombres de las columnas (ID_ENA2025_CUEST_UP, EC110, PP111_01, etc.).
- Tipos Java mapeados desde los tipos de metadatos (por ejemplo `Long`, `Double`, `String`).

- Getters, setters y un `toString()` básico.

```java
package mx.inegi.generated;

public class TR_ENA2025_CUEST_UP1 {

    private Long ID_ENA2025_CUEST_UP;
    private Long EC110;
    private Long EC111_01;
    private Long EC111_02;
    private Long EC111_03;
    private Long EC111_04;
    private Long EC111_99;
    private String EC111_991_CVE;
    private Long PP111_021;
    private Long PP111_023;
    private Long PP111_024;
    private Long PP111_026;
    private Long PP111_027;
    private Long PP111_028;
    private Long PP111_0211;
    private Long PP111_161;
    private Long PP111_162;
    private Long PP111_01;
```

6. Paso a paso – JSON a documentos RAG y chat de metadatos

6.1 Generar documentos de tabla y columnas

A partir del JSON de metadatos, se pueden generar documentos de texto.

Comando:

```
venv\Scripts\activate
python -m python.metadata_to_rag_docs \
  --json output/metadata_TR_ENA2025_CUEST_UP1.json \
  --out output/docs_TR_ENA2025_CUEST_UP1.json
```

(todo en una línea) Salida esperada:

```
Resumen de documentos generados:
  Total: 45
  Documentos de tabla: 1
  Documentos de columna: 44

Ejemplos de títulos:
  - Tabla TR_ENA2025_CUEST_UP1
  - Columna ID_ENA2025_CUEST_UP en TR_ENA2025_CUEST_UP1
  - Columna EC110 en TR_ENA2025_CUEST_UP1
```

Estos documentos se guardan en `output/docs_TR_ENA2025_CUEST_UP1.json` y son la base del esquema de recuperación.

6.2 Ejecutar el chat de metadatos con un modelo en español

El chat se ejecuta con `llm_stub/chat_demo.py`, que combina:

- Carga de metadatos JSON.
- Construcción de documentos para tabla y columnas.
- Recuperador simple por tabla, columna o palabra clave.
- Cliente de modelo de lenguaje.

Modelo usado en este prototipo:

- `LenguajeNaturalAI/leniachat-qwen2-1.5B-v0`
  Es un modelo muy básico conversacional en español, basado en Qwen2 1.5B, orientado a chat e instrucciones y distribuido bajo licencia Apache 2.0, diseñado específicamente para texto y asistentes en español.
  Su tamaño permite ejecutarlo en CPU o GPU modesta sin requerir recursos extremos. Pagina de leniachat-qwen2 en HuggingFace

Comando para ejecutar el chat con ese modelo:

```
venv\Scripts\activate
python -m llm_stub.chat_demo
  --json output/metadata_TR_ENA2025_CUEST_UP1.json
  --model LenguajeNaturalAI/leniachat-qwen2-1.5B-v0
```

(igual que los pasados comandos esto va en una sola línea en tu terminal al momento de correrlo)

El programa:

1. Carga el JSON de metadatos y construye los documentos.
2. Inicializa el recuperador.
3. Inicializa el modelo de lenguaje vía `transformers`.
4. Abre un ciclo de preguntas en consola.

Ejemplo de interacción real:

```
============================================================
Chat de metadatos INEGI
============================================================
Escribe 'salir' o 'exit' para terminar

Tu pregunta: donde se usa la variable PP112_991_CVE y cual es su nivel de
```

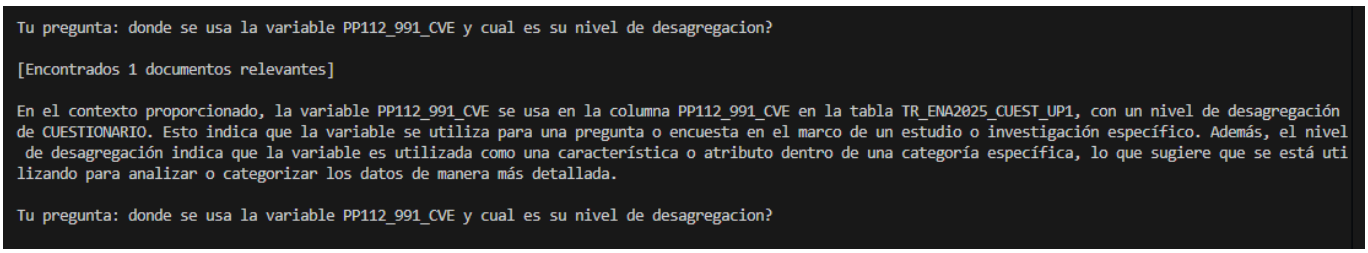
```
desagregacion?

[Encontrados 1 documentos relevantes]

En el contexto proporcionado, la variable PP112_991_CVE se usa en la columna
PP112_991_CVE
en la tabla TR_ENA2025_CUEST_UP1, con un nivel de desagregación de CUESTIONARIO...
```

Ejemplos de preguntas útiles para probar:

- `¿qué significa PP111_01?`
- `donde se usa la variable PP111_01?`
- `cual es el nivel de desagregacion de EC110?`
- `que columnas tiene la tabla TR_ENA2025_CUEST_UP1?`

Si no se encuentra contexto relevante en los documentos, el sistema indica que no hay documentos relacionados en los metadatos.

Si se ejecuta el chat **sin** parámetro `--model`, el cliente utiliza el modo simulado definido en el código, que genera respuestas breves basadas únicamente en el texto de los documentos, sin llamar a un modelo externo. Este modo sirve para pruebas rápidas sin descargar modelos.




```
Nota sobre la calidad del ejemplo de respuesta
El ejemplo del screenshot anterior tiende a sobreexplicar y a repetir partes de la
pregunta.
Esto no es un error del flujo de metadatos, sino una limitación de tres factores:

- El modelo utilizado es pequeño y genérico.
- El recuperador actual es básico y se apoya solo en coincidencias de texto.
- No se ha realizado ningún entrenamiento específico con preguntas reales del
dominio.

Con un RAG más sólido y un modelo más grande o ajustado al contexto de una
encuesta nacional,
las respuestas pueden ser más precisas y directas.
Este prototipo solo demuestra que el sistema ya recupera el metadato correcto y
puede generar una respuesta basada en él.
La optimización de estilo, síntesis y claridad corresponde a etapas posteriores.
```

6.3 Justificación del modelo elegido y relación con otros modelos

Se eligió **LenguajeNaturalAI/leniachat-qwen2-1.5B-v0** porque ofrece buena comprensión del español, es ligero y puede ejecutarse en CPU o GPU de gama media sin infraestructura especializada. Además, la base Qwen2 proporciona un rendimiento consistente en tareas de diálogo.

En equipos con más capacidad de cómputo se pueden considerar modelos más grandes, por ejemplo:

- `meta-llama/Llama-3.1-8B-Instruct`
  Modelo multilingüe (incluye español) con 8B de parámetros, orientado a diálogo e instrucciones. Requiere una GPU más grande pero ofrece mejor comprensión y generación. *(Info. en Huggingface)*

- `Qwen/Qwen2.5-7B-Instruct`
  Modelo de 7B parámetros, multilingüe, con buen rendimiento en tareas de razonamiento, código y generación en varios idiomas, incluido español. *(Info. en Hugginface)*

Este *README* se centra en *`leniachat-qwen2-1.5B-v0`* por ser una opción práctica para pruebas iniciales. Las alternativas de mayor escala y sus beneficios se detallan en el punto 9.

7. Prueba guiada del flujo completo

Este apartado describe una ruta mínima para que cualquier persona pueda ejecutar todo el prototipo de inicio a fin, desde el Excel hasta la clase Java y el chat de metadatos.

7.1 Preparación del entorno

Desde la raíz del proyecto:

```
cd inegi-meta-llm
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

Verificar que Python puede importar los módulos:

```
python -c "import pandas, transformers; print('OK')"
```

7.2 Generar metadatos (JSON) desde una hoja de Excel

(Ya se debe estar dentro del entorno virtual, si no, se tiene que correr de nuevo el comando: `venv\Scripts\activate`)

Una vez dentro del venv:

```
python -m python.cli_generate_metadata`
```

Pasos en consola:

1. El sistema mostrará los archivos `.xlsx` encontrados en la carpeta `excel`.

2. Seleccionar el número correspondiente al archivo de interés.
3. El sistema detectará las tablas en la primera hoja y pedirá elegir una.
4. Al finalizar se generará un archivo JSON en `output/`, por ejemplo:
    - `output/metadata_TR_ENA2025_CUEST_UP1.json`

Debe aparecer un mensaje indicando que la validación de metadatos se realizó de forma exitosa. Ejemplo:

```
Se encontraron varios archivos Excel. Selecciona uno:
  1. UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx
  2. ~$UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx

Ingresa el número del archivo a usar: 1

Se encontraron 18 tablas en la hoja 'COMPLETO'. Selecciona una:
  1. TRS_ENA2025_CONTROL_CUPA
  2. TR_ENA2025_ABONO
  3. TR_ENA2025_APROTEGIDA
  4. TR_ENA2025_CABIERTO
  5. TR_ENA2025_CUEST_UP1
  6. TR_ENA2025_CUEST_UP2
  7. TR_ENA2025_CUEST_UP3
  8. TR_ENA2025_CUEST_UP4
  9. TR_ENA2025_CUEST_UP5
  10. TR_ENA2025_FERTILIZANTE
  11. TR_ENA2025_NUTRICION
  12. TR_ENA2025_UNIDADES_PRODUCCION
  13. TR_ENA2025_VAVES
  14. TR_ENA2025_VBOVINO
  15. TR_ENA2025_VCAPRINO
  16. TR_ENA2025_VOVINO
  17. TR_ENA2025_VPORCINO
  18. VARIABLE GENERADA

Ingresa el número de la tabla a procesar: 2

Procesando tabla 'TR_ENA2025_ABONO' desde UNIDADES_OBSERVACIÓN_ENA2025_08082025.xlsx...
  Hoja: COMPLETO
  Columnas encontradas: 7
  Metadatos guardados en: output\metadata_TR_ENA2025_ABONO.json

Validando metadatos generados...
✓ Validación exitosa

Archivo generado correctamente: output\metadata_TR_ENA2025_ABONO.json
```

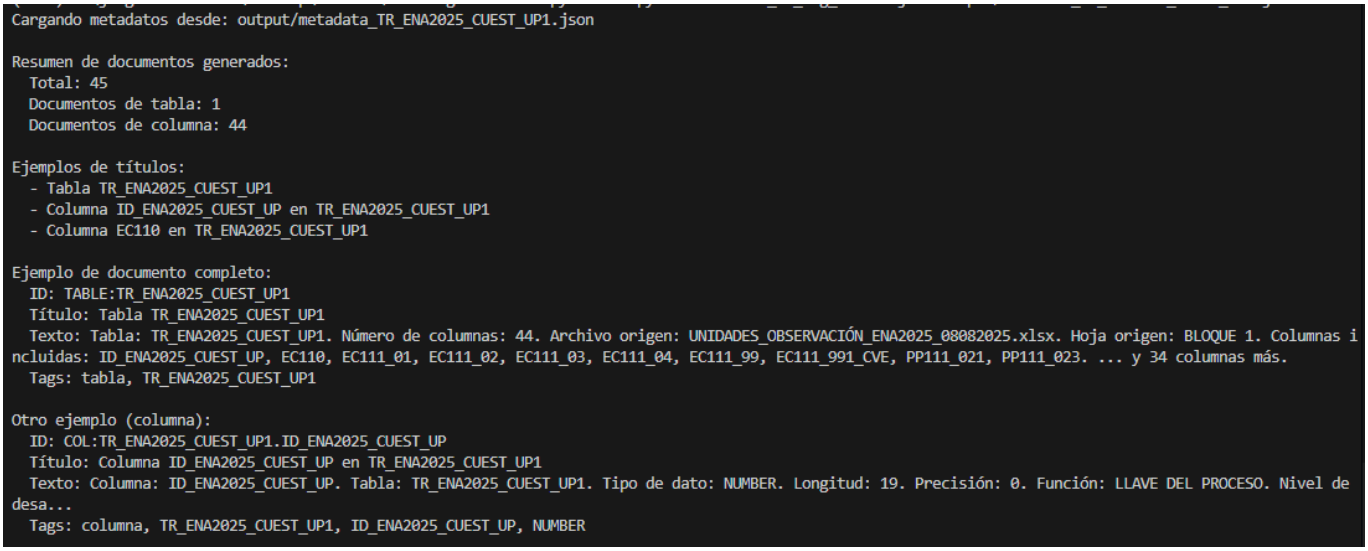
7.3 Generar documentos RAG para esa tabla

```
python -m python.metadata_to_rag_docs
  --json output/metadata_TR_ENA2025_CUEST_UP1.json
```

Esto construye:

- Un documento de tabla.
- Un documento por columna.

El resumen en consola debe indicar el número total de documentos.



7.4 Probar el chat de metadatos con el modelo en español

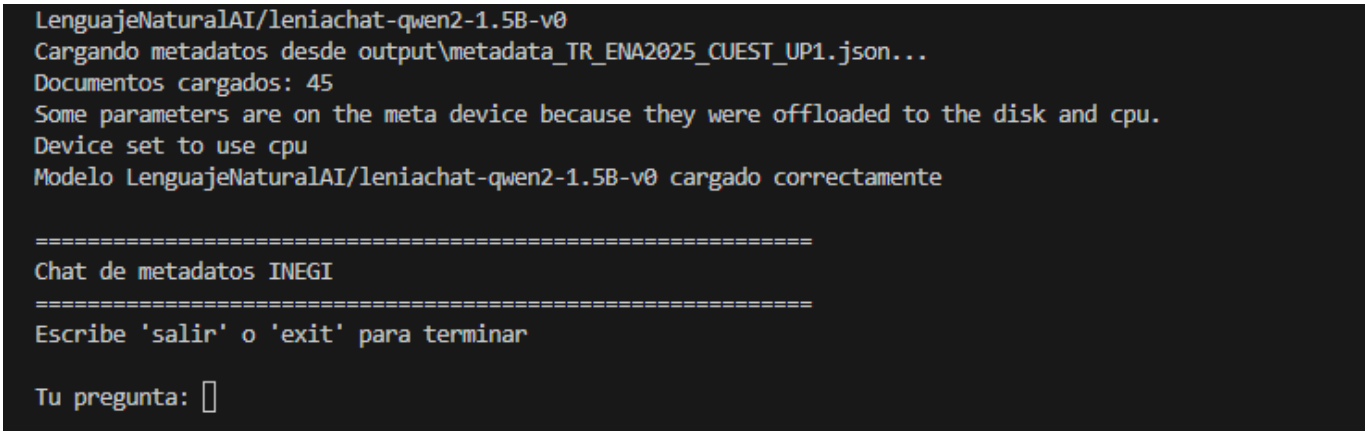
```
python -m llm_stub.chat_demo
  --json output/metadata_TR_ENA2025_CUEST_UP1.json
  --model LenguajeNaturalAI/leniachat-qwen2-1.5B-v0
```

En la consola se mostrará un encabezado de chat y se podrá escribir preguntas como:

- `¿qué significa PP111_01?`
- `donde se usa la variable PP111_01`
- `cual es el nivel de desagregacion de PP112_991_CVE`
- `que columnas tiene la tabla TR_ENA2025_CUEST_UP1`

El sistema indicará cuántos documentos encontró y mostrará una respuesta basada en el texto de los metadatos.

Si el modelo no está descargado aún, `transformers` lo descargará desde Hugging Face al primer uso.



7.5 Generar la clase Java desde los metadatos

Ir a la carpeta Java:

```
cd java
mvn compile
```

Una vez compilado, ejecutar:

```
mvn exec:java -Dexec.args="../output/metadata_TR_ENA2025_CUEST_UP1.json
mx.inegi.generated"
```

El programa:

1. Lee el archivo JSON con `JsonMetadataLoader`.
2. Construye un objeto `TableMetadata`.
3. Genera una clase Java bajo el paquete indicado, por ejemplo:
    - `src/generated/mx/inegi/generated/TR_ENA2025_CUEST_UP1.java`

Con esto se puede revisar una clase de dominio generada directamente desde la hoja de diseño.

7.6 Pruebas automatizadas

Para validar el comportamiento básico con `pytest`:

```
cd inegi-meta-llm
venv\Scripts\activate
pytest tests
```

Las pruebas incluidas verifican:

- Que desde un Excel de ejemplo se puede construir un `TableMetadata` válido.
- Que a partir de un JSON se generan documentos de tabla y columnas.
- Que el recuperador simple devuelve documentos relevantes para búsquedas por tabla, columna y palabra clave.

Así se vería la consola con los tests superados:

```
========================================= test session starts =========================================
platform win32 -- Python 3.13.6, pytest-9.0.1, pluggy-1.6.0
rootdir: D:\jorge.villalobos\Desktop\SCRIPTS\5 - inegi-meta-llm
collected 16 items

tests\test_excel_to_metadata.py ....                                                          [ 25%]
tests\test_metadata_to_rag_docs.py .....                                                      [ 56%]
tests\test_simple_retriever.py .......                                                        [100%]

========================================= 16 passed in 2.74s ==========================================
```

8. Uso con múltiples tablas y cuestionarios

El flujo actual trabaja sobre una tabla a la vez, representada por un archivo JSON de metadatos. Para escalar a múltiples cuestionarios y encuestas se sugiere:

1. Mantener un JSON por tabla, organizado por encuesta y año, por ejemplo:
    - `output/ENA2025/metadata_TR_ENA2025_CUEST_UP1.json`
    - `output/ENA2025/metadata_TR_ENA2025_CUEST_UP2.json`
    - `output/ENA2025/metadata_TR_ENA2025_CUEST_UP3.json`
2. Para cada JSON generar su archivo de documentos:
    - `output/ENA2025/docs_TR_ENA2025_CUEST_UP1.json`, etcétera.
3. En una fase siguiente, implementar un módulo que:
    - Recorra todas las carpetas de metadatos.
    - Convierta cada JSON en documentos de texto.
    - Agregue etiquetas adicionales: encuesta, año, bloque, versión.
    - Construya un catálogo consolidado (por ejemplo un único JSON grande o una base de datos).
4. Ajustar el chat para trabajar contra ese catálogo consolidado, en lugar de un único archivo, cambiando:
    - De `--json archivo.json`
    - A un parámetro del tipo `--data-root output/ENA2025/` y un cargador que una todos los documentos.

El esquema de RAG se mantiene igual. Lo que crece es el volumen de documentos administrados y la lógica para agruparlos y filtrarlos.

9. Modelos de lenguaje y escalamiento futuro

9.1 Modelo actual del prototipo

El prototipo utiliza **`LenguajeNaturalAI/leniachat-qwen2-1.5B-v0`** como modelo por defecto en las pruebas con modelo real porque:

- Está desarrollado específicamente para aplicaciones de chat y generación de texto en español.
- Se basa en Qwen2 1.5B, que ofrece buen rendimiento con un número moderado de parámetros, lo que lo vuelve viable en equipos sin GPU de gran capacidad.
- Utiliza una licencia Apache 2.0, adecuada para prototipos y proyectos internos.

Para la mayoría de los usos de prueba y demostración es suficiente.

9.2 Opciones cuando se cuenta con GPU más potente

Si se dispone de una GPU o CPU con más memoria y se busca mayor calidad de respuesta, se pueden considerar modelos como:

- **`meta-llama/Llama-3.1-8B-Instruct`** *(Info. en Huggingface)*

  Modelo multilingüe (incluye español) optimizado para diálogo, con 8B de parámetros y contexto largo. Ofrece mejor comprensión y generación, a costa de un mayor consumo de memoria.

- `Qwen/Qwen2.5-7B-Instruct` *(Info. en Huggingface)*

  Modelo de 7B parámetros, multilingüe, con buen desempeño en comprensión, razonamiento y generación estructurada, con soporte explícito para español.

La arquitectura del prototipo (RAG) permite cambiar de modelo sin modificar el formato de los metadatos JSON ni la lógica de generación de clases Java. Solo hay que ajustar el parámetro `--model` y revisar los límites de memoria y tiempo de respuesta.

9.3 Entrenamiento y adaptación futura

Para una fase de escalamiento, se sugiere:

1. Consolidar un repositorio de documentos:
   - Metadatos JSON enriquecidos.
   - Manuales de conceptos.
   - Reglas de validación.
   - Catálogos y descripciones de variables.
2. Implementar un índice vectorial:
   - Usar un modelo de embeddings multilingüe.
   - Guardar vectores y metadatos en un índice especializado.
   - Integrarlo con el flujo de recuperación actual.
3. Recopilar registros de preguntas reales de analistas y validadores, y generar respuestas validadas. Estas parejas pregunta–respuesta pueden servir para ajustar mas fino a un modelo base mediante técnicas como LoRA o QLoRA, que permiten ajustar modelos grandes de forma eficiente sobre hardware limitado.

La lógica de generación de clases y metadatos no depende del modelo de lenguaje. Por lo tanto, el trabajo invertido en la parte Excel -> JSON -> Java se conserva aunque en el futuro se cambie de modelo o se pase a una arquitectura de RAG más compleja.

10. Resumen rápido de ejecución

Secuencia mínima para recorrer el flujo completo:

1. Activar entorno Python:

```
cd inegi-meta-llm
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

2. Generar metadatos desde Excel:

```
python -m python.cli_generate_metadata
```

3. Generar documentos RAG :

```
python -m python.metadata_to_rag_docs
--json output/metadata_TR_ENA2025_CUEST_UP1.json
```

4. Probar el chat de metadatos:

```
python -m llm_stub.chat_demo
--json output/metadata_TR_ENA2025_CUEST_UP1.json
--model LenguajeNaturalAI/leniachat-qwen2-1.5B-v0
```

5. Generar la clase Java:

```
cd java
mvn compile
mvn exec:java -Dexec.args="../output/metadata_TR_ENA2025_CUEST_UP1.json
mx.inegi.generated"
```

Con estos pasos se demuestra de manera concreta que:

- Las hojas de diseño de la encuesta se convierten en metadatos estructurados.
- Los metadatos se usan para generar clases Java de dominio.
- Es posible consultar tablas y variables mediante un chat que combina recuperación y modelos de lenguaje.