IA EN EL INEGI:

PROTOTIPO DE INFRAESTRUCTURA DE METADATOS Y ASISTENTE LLM CON RAG (Retrieval-Augmented Generation)

Jorge Luis Villalobos Medrano

Diciembre de 2025

Contenido

1. Contexto y objetivo del proyecto

En ciertos procesos actuales de proyectos del instituto como la ENA, una parte importante de la validación de información se apoya en plantillas de Excel (unidades de observación, catálogos, estructuras de datos, etc.), que son convertidos en archivos de clases de Java que implementan reglas de validación.

El flujo actual ya está resuelto con automatizaciones que convierten plantillas de Excel en un JSON de metadatos y a partir de ahí, generan las clases Java y sus validadores.

Esto funciona, pero es rígido, cualquier cambio en el diseño de Excel, en los catálogos o en las reglas obliga a mantener plantillas, scripts y estructuras a mano.

La intención no es rehacer ese flujo, sino, explorar técnicas de IA que nos permitan crear una capa de conocimiento unificada que:

- Entienda qué significan las variables.
- Sepa qué reglas se aplican en cada bloque.
- Conozca la evolución de los ejercicios entre proyectos/encuestas (ENA, CE, etc.).
- Sea consultable de forma centralizada (dentro del instituto) por un asistente inteligente.

2. Giro del proyecto: pasar de flujo puntual a infraestructura de conocimiento con “RAG”

Antes de explicar el giro del proyecto, es necesario aclarar qué es *RAG (Retrieval-Augmented Generation).*

RAG es una técnica donde un modelo de lenguaje no inventa respuestas, consulta primero una base de conocimiento actualizada, recupera la información relevante y luego genera la respuesta basándose en esos documentos. Es la diferencia entre

un modelo entrenado “que recuerda cosas” y uno que “va a buscarlas y luego responde”.

Esto permite mantener un sistema siempre alineado a reglas, metadatos y estructuras vigentes sin tener que reentrenar el modelo cada vez que cambia algo, si no, cada encuesta o censo se tendría que reentrenar al modelo.

El proyecto deja de ser solo una mejora puntual a un flujo ya funcional y se plantea como un piloto técnico para algo más amplio, usar la ENA 2025 como dominio controlado donde se definan metadatos en JSON enriquecidos a partir de las plantillas de Excel, se genere automáticamente el código Java que el equipo necesita, y al mismo tiempo se construya la base de un asistente local con LLM y RAG.

En la práctica, esto significa que cada ajuste al diseño en Excel no solo se refleja en las clases Java, sino que también alimenta un repositorio de conocimiento estructurado sobre tablas, campos y reglas de la ENA 2025. Ese repositorio podrá ser consultado por un chat interno, y más adelante reutilizado en otros ejercicios del instituto sin volver a empezar desde cero.

Aquí es donde entra la tecnología RAG (*Retrieval-Augmented Generation*), en lugar de entrenar un modelo una sola vez y dejarlo estático, se construye un índice de conocimiento actualizable:

- Metadatos de todas las Unidades.
- Documentos de requerimiento.
- Manuales de levantamiento y validación.
- Código y ejemplos relevantes.

De esta manera el LLM no se inventará o alucinará las respuestas, en cambio consultará ese índice cada vez que responde. Cuando lleguen a cambiar los cuestionarios, variables o reglas, no hay que reentrenar el modelo, basta con actualizar el conjunto de documentos y metadatos indexados.

3. Flujo actual del proceso base (ya existente)

1. **Plantillas de Excel**

   Cada bloque / unidad de observación se define en hojas de Excel con nombre de tabla, campo, tipo, longitud, función, nivel de desagregación, entre otras variables.

2. **JSON de metadatos mínimos**

   A partir de esas hojas se genera un JSON sencillo por bloque (nombre del bloque y lista de campos con su tipo básico: integer, string, etc.).

3. **Clases Java y procesos**

   Ese JSON se usa para generar o parametrizar clases de Java que luego participan en los procesos de la ENA, el flujo ya está implementado y operativo actualmente.

3.1. Limitaciones

El JSON solo guarda tipos básicos, no documenta rangos, dominios, nulabilidad explícita, catálogos ni variables de vínculo directo con las preguntas del cuestionario.

La información real de las variables está repartida entre: archivos de Excel, tablas en las bases de datos, cuestionarios, código en Java, SQL y conocimiento del equipo.

No hay una representación estándar que sirva a la vez como fuente única de verdad de cada bloque procesado, ni como insumo directo para indexar en un esquema de RAG y alimentar un LLM local.

El nuevo proyecto se apoya en este flujo, pero su objetivo es concentrar todo ese conocimiento en metadatos enriquecidos y hacerlos aprovechables por modelos IA con RAG.

4. Propuesta de arquitectura con IA

La arquitectura propuesta se plantea con dos salidas principales a partir de las plantillas de Excel.

1. Generar de forma automática las clases Java necesarias para los procesos de la encuesta o proyecto en curso.
2. Exponer un asistente tipo chat, basado en un LLM local con RAG, capaz de:
    - Consultar el mismo metadato que se usa para generar las clases.
    - Responder dudas sobre tablas, campos y reglas.
    - Servir como punto de entrada para otros proyectos similares.

El punto de partida será siempre el mismo: las hojas de Excel que definen datos como el nombre de tabla, campo, tipo, longitud, función o nivel de desagregación.

4.1. Componentes principales

4.1.1. Ingesta de plantillas de Excel

- Lee hojas de diseño base.
- Extrae por fila: nombres relevantes, tipo de variables, longitud y precisión, función (llave, consulta, modificable, etc.).

Resultado: una estructura intermedia en memoria (o 1er JSON simple) por tabla.

4.1.2. Motor de metadatos (reglas fijas + LLM)

Toma esta estructura intermedia y genera una propuesta de metadatos enriquecidos (datos extra útiles) por tabla, con:

- nombre descriptivo (cuando se pueda inferir),
- tipo lógico (entero, decimal, clave, sí/no, etc.),

- nulabilidad (derivada de función y uso),
- reglas básicas (por ejemplo: “superficies y cantidades no negativas”, “sí/no codificado 1-9”, cuando se detecta el patrón en el nombre).

Aquí participa el modelo de IA:

- Reconoce patrones en nombres de campos (*_01, *_UM, *_OTRO).
- Propone descripciones cortas de los campos.
- Sugiere si algo parece cantidad, indicador, clave de catálogo, etc.

El resultado se expresa ya en el formato JSON de metadatos definido en la siguiente sección.

4.1.3. Generador de JSON de metadatos

- Recibe la propuesta del motor, y la ajusta al esquema estándar.
- Completa campos obligatorios *(tableName, columns, domain, version, etc.).*
- Normaliza tipos *(INTEGER, DECIMAL, STRING)*.
- Marca qué columnas parecen llaves (apoyado en función y nombre).
- Aplica validaciones estructurales, y para lo que no cumple se marca para corrección manual.

4.1.4. Generador de clases Java

A partir del JSON de metadatos, genera automáticamente una clase Java por tabla (con campos, tipos Java, getters/setters), y anotaciones de validación básicas, cuando sea posible (@NotNull, etc.).

El sistema usa plantillas, haciendo que la salida sea determinista y reproducible.

Este componente es el que resuelve el objetivo operativo inmediato, pasar del Excel a las clases Java sin codificación manual.

4.1.5. Constructor del índice RAG

Toma como insumo los JSON de metadatos enriquecidos generados a partir de los Excel, y cuando existan, revisa documentos adicionales que el equipo cargue (requerimientos, manuales, notas técnicas, cuestionarios).

Los transforma en documentos indexables por encuesta, bloque, tabla y campo.

El índice se puede actualizar sin reentrenar el modelo, basta con incorporar nuevos metadatos o documentos.

4.1.6. Servicio LLM local con RAG

El mismo proyecto expone un chat interno donde el usuario puede preguntar, por ejemplo:

- "¿Qué campos tiene la tabla TR_CABIERTO y para qué se usa cada uno?"
- "Genera de nuevo la clase Java para esta tabla con los últimos cambios del Excel"
- "¿Qué variables no son numéricas en el catálogo en este bloque?"

El flujo sería:

1. El usuario escribe en el chat.
2. El módulo de recuperación busca en el índice los metadatos y documentos relevantes.
3. El LLM responde usando ese contexto: puede devolver texto explicativo, o incluso la plantilla de clase Java correspondiente, generada a partir del JSON.

4.2. Qué usa IA y qué es determinista

Uso de IA (LLM):

- Interpretación de nombres de campos y patrones en el Excel.
- Propuesta de tipo lógico y reglas básicas cuando el patrón es claro.
- Generación de descripciones cortas de campos y tablas.
- Respuestas del chat, apoyadas en el índice RAG.
- Generación “bajo demanda” de borradores de clases Java a partir de los metadatos indexados.

Código determinista clásico:

- Lectura de Excel y construcción de la estructura intermedia.
- Normalización de tipos y construcción del JSON según el esquema acordado.
- Generación de clases Java a partir del JSON.
- Validaciones de estructura del JSON y compilación de código.
- Construcción y actualización del índice RAG.

4.3. Flujo completo lógico resumido

1. El usuario carga una plantilla de Excel de diseño de tablas/bloques.
2. El módulo de ingesta la lee y arma una estructura técnica por tabla.
3. El motor de metadatos, apoyado en reglas fijas y el LLM, propone un JSON de metadatos enriquecido.
4. El generador de JSON valida y normaliza esa propuesta.

5. A partir de ese JSON, el generador de código produce la clase Java correspondiente, y el constructor del índice RAG incorpora el metadato al repositorio de conocimiento.
6. El servicio de chat LLM consulta ese índice para:
    - explicar tablas y campos.
    - sugerir cambios.
    - devolver clases Java actualizadas cuando se le solicite.

Esta arquitectura no solo se plantea a nivel conceptual, sino que ya cuenta con un prototipo implementado que materializa el flujo de trabajo propuesto. El detalle del prototipo y su repositorio de código se presenta en la sección 7.

5. Modelo de metadatos (JSON) propuesto

5.1. Idea general

El modelo de metadatos se define como un JSON estándar que, para cada tabla descrita en Excel, concentre información básica de la tabla (nombre, descripción, encuesta, versión), lista de columnas con tipo lógico, longitud, nulabilidad y reglas simples, referencia opcional a catálogos o preguntas, y metadatos técnicos (origen, fecha de generación, autor).

Este JSON es el contrato único entre el generador de clases Java, y el índice de conocimiento que usará el LLM.

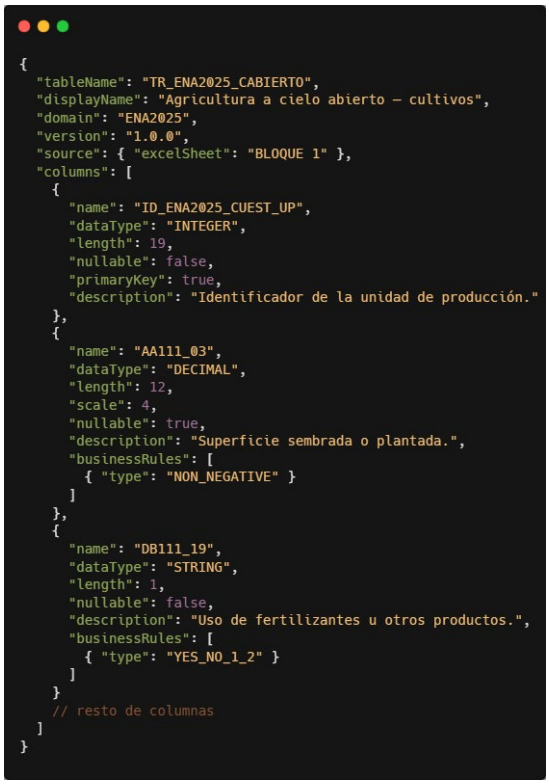
5.2. Estructura básica

```
{
  "tableName": "TR_ENA2025_CABIERTO",
  "displayName": "Agricultura a cielo abierto – cultivos",
  "domain": "ENA2025",
  "version": "1.0.0",
  "source": {
    "excelSheet": "BLOQUE 1"
  },
  "columns": [
    {
      "name": "ID_ENA2025_CUEST_UP",
      "dataType": "INTEGER",
      "length": 19,
      "nullable": false
    },
    {
      "name": "AA111_03",
      "dataType": "DECIMAL",
      "length": 12,
      "scale": 4,
      "nullable": true
    }
    // más columnas
  ]
  // catálogos y metadatos adicionales
}
```

Los campos como *displayName*, *description*, *businessRules* o *questionRef* se agregan como extra cuando el motor de metadatos (reglas + LLM) puede inferirlos o cuando el equipo los define explícitamente.

5.3. Ejemplo breve de metadatos enriquecidos

A modo de referencia, un ejemplo ligeramente más completo podría verse así:



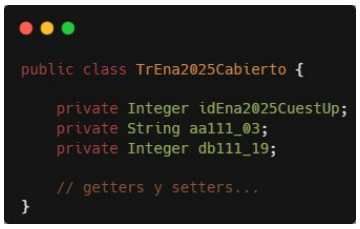
```
{
  "tableName": "TR_ENA2025_CABIERTO",
  "displayName": "Agricultura a cielo abierto – cultivos",
  "domain": "ENA2025",
  "version": "1.0.0",
  "source": { "excelSheet": "BLOQUE 1" },
  "columns": [
    {
      "name": "ID_ENA2025_CUEST_UP",
      "dataType": "INTEGER",
      "length": 19,
      "nullable": false,
      "primaryKey": true,
      "description": "Identificador de la unidad de producción."
    },
    {
      "name": "AA111_03",
      "dataType": "DECIMAL",
      "length": 12,
      "scale": 4,
      "nullable": true,
      "description": "Superficie sembrada o plantada.",
      "businessRules": [
        { "type": "NON_NEGATIVE" }
      ]
    },
    {
      "name": "DB111_19",
      "dataType": "STRING",
      "length": 1,
      "nullable": false,
      "description": "Uso de fertilizantes u otros productos.",
      "businessRules": [
        { "type": "YES_NO_1_2" }
      ]
    }
    // resto de columnas
  ]
}
```

6. Generación de clases Java y validadores

Una vez que el motor de metadatos produce el JSON enriquecido de una tabla, el generador de código recorre la lista de columnas y aplica plantillas predefinidas para construir la clase Java correspondiente.

De forma simplificada, para cada *columns[i]* del JSON:

- Se mapea dataType -> tipo Java (INTEGER -> Integer, DECIMAL -> BigDecimal, etc.).
- Se crea un campo privado.
- Se generan el getter y el setter.
- Si hay reglas sencillas (nullable = false, NON_NEGATIVE, YES_NO), se agregan anotaciones de validación cuando aplica.



La idea es que la estructura viene dictada por el JSON, no por código escrito a mano.

6.1. Incorporación de reglas de validación

El JSON permite también incorporar reglas mínimas de validación, que pueden expresarse en Java de dos maneras:

6.1.1. Anotaciones estándar

Reglas muy simples se pueden traducir a anotaciones de Bean Validation.

Ejemplos típicos:

REGLA	***VALIDATION***
nullable = false	*@NotNull*
NON_NEGATIVE	*@Min(0)*
longitudes de texto	*@Size(max = N)*

6.1.2. Validadores auxiliares

Para reglas que no encajan en una anotación estándar, el sistema puede generar métodos o utilidades de validación apoyadas en los metadatos, como una clase genérica que recorra las columnas y verifique los campos que agrego la IA como “businessRules”, o validadores específicos por tabla, si se justifica.

Lo importante es que el punto de entrada sea siempre el JSON, de modo que si cambia la estructura de la tabla, o se incorporan nuevas reglas, el generador pueda volver a crear la clase sin reescribir todo.

6.2. Relación con el chat LLM

El LLM no compila ni despliega código, pero puede leer los metadatos y mostrar la estructura de la clase, sugerir cómo quedaría una clase para una nueva tabla, o devolver un bloque de código Java coherente con el JSON actual.

Así, la generación automática se hace por el generador determinista, y el LLM sirve como interfaz para explorar y entender esas clases sin abrir directamente los archivos o el código fuente.

7. Rol de la IA/LLM en el flujo

7.1. Puntos específicos donde interviene el LLM

La IA entra en dos frentes muy claros:

1. Dentro del flujo técnico que parte del Excel, a partir de las hojas de diseño, el modelo ayuda a interpretar nombres de campos, detectar patrones (cantidades, unidades, tipos especiales de respuesta, claves, etc.) y proponer tipos lógicos, descripciones y reglas básicas.

   Esas propuestas se traducen a JSON de metadatos, pero siempre pasan por un esquema fijo y por revisión humana cuando afectan algo delicado (llaves, reglas de validación relevantes, etc.). La IA sugiere, el JSON y el código se definen de forma determinista.

2. En la capa de consulta, Una vez que los metadatos JSON y otros documentos del proyecto se indexan, el LLM local funciona como chat interno: responde qué tablas existen, qué campos tiene un bloque, qué significa una variable, o cómo quedaría la clase Java actualizada para cierta tabla.

   El LLM podría responder a solicitudes como:

   - “Genera la clase Java de esta tabla con los tipos actuales del diseño.”
   - “¿Qué columnas se usan como llaves en esta unidad de observación?”

   Así como también podría dar contexto entre proyectos, cuando se agreguen otros ejercicios, el mismo LLM debería poder:

   - Cambiar de contexto entre encuestas.
   - Comparar definiciones.
   - Señalar diferencias entre versiones.

La gracia del RAG es que, al cambiar un Excel o un documento, basta actualizar el índice para que las respuestas reflejen la nueva información, sin reentrenar todo el modelo.

7.2. Riesgos y controles

Los riesgos principales son:

Respuestas inventadas o interpretaciones incorrectas si no hay suficiente información, y posibles inconsistencias si Excel, metadatos y documentos no están alineados, por eso se plantean tres controles básicos:

1. Usar un esquema JSON estricto.
2. Mantener el generador de código como pieza determinista.

3. Exigir revisión humana en cambios de estructura y reglas que tengan impacto operativo.

El LLM se usa como asistente, no como fuente de verdad.

8. Prototipo implementado y repositorio de código

Como parte de este proyecto se desarrolló un prototipo funcional que implementa el flujo descrito en este documento: lectura de plantillas de diseño en Excel, generación de metadatos JSON, generación automática de clases Java y un chat de consulta de metadatos apoyado en un modelo de lenguaje con RAG simplificado.

El código se encuentra publicado en el repositorio público:

https://github.com/jorge-inegi/inegi-meta-llm

Por tratarse de información interna del instituto, el repositorio solo incluye el código fuente, scripts de ejemplo y documentación técnica. Las plantillas reales de diseño de la ENA 2025 y los metadatos derivados no se versionan en GitHub y deben incorporarse de forma local por quien ejecute el prototipo.

Si en algún momento se considera necesario migrar este trabajo a un entorno privado o a infraestructura interna del INEGI, el contenido del repositorio puede clonarse o copiarse y complementarse con los archivos de diseño reales de cada proyecto.