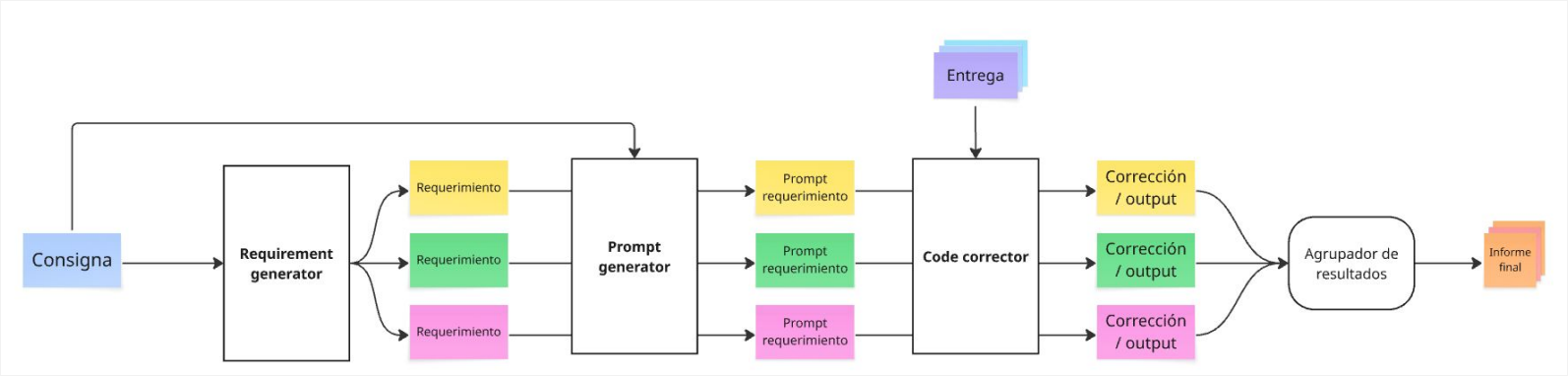
# Deep Learning
# TP 2 - RAG
# JADE

62494 - Wischñevsky, David
62495 - Vilamowski, Abril
62533 - Liu, Jonathan

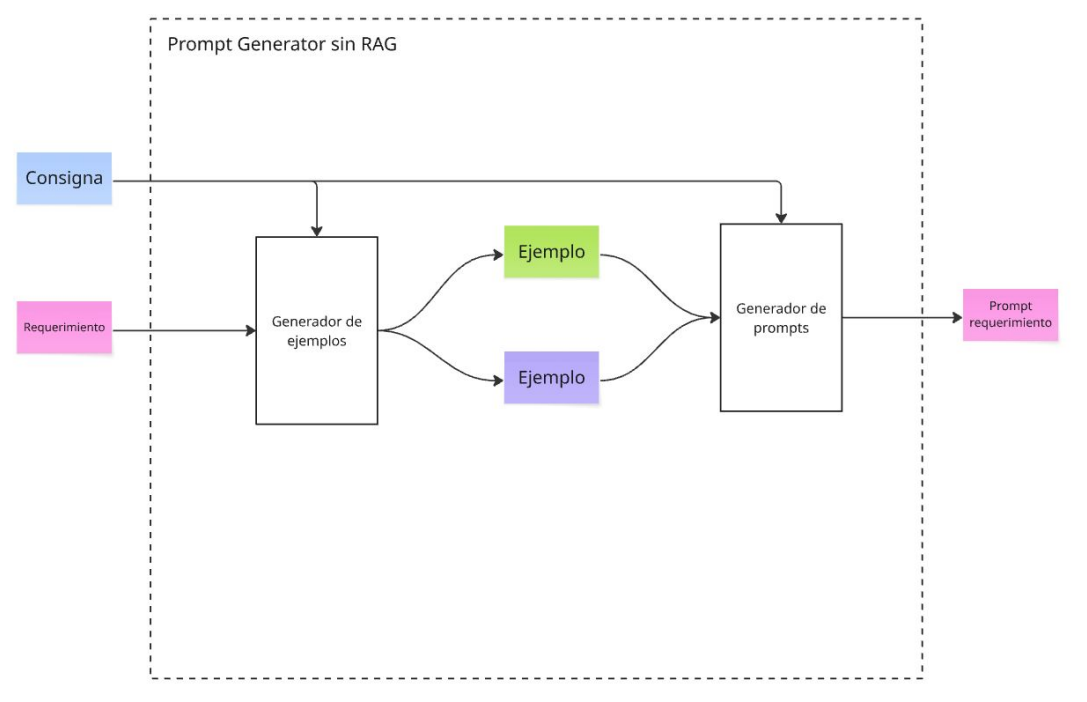
# Problemática

- La enseñanza introductoria de programación enfrenta problemas por la gran cantidad de estudiantes y la corrección manual intensiva.
- Las herramientas actuales no permiten evaluar errores conceptuales cuando el código no ejecuta.
- No se puede confiar únicamente en el corpus de conocimiento de los modelos para hacer una corrección acotada a un curso específico.

# Arquitectura general del Pipeline



# Generador de Prompts sin RAG



# Generación de Ejemplos

Para cada venta, se debe elegir al azar un producto del inventario actual y vender entre 1 y 5 unidades, también al azar.

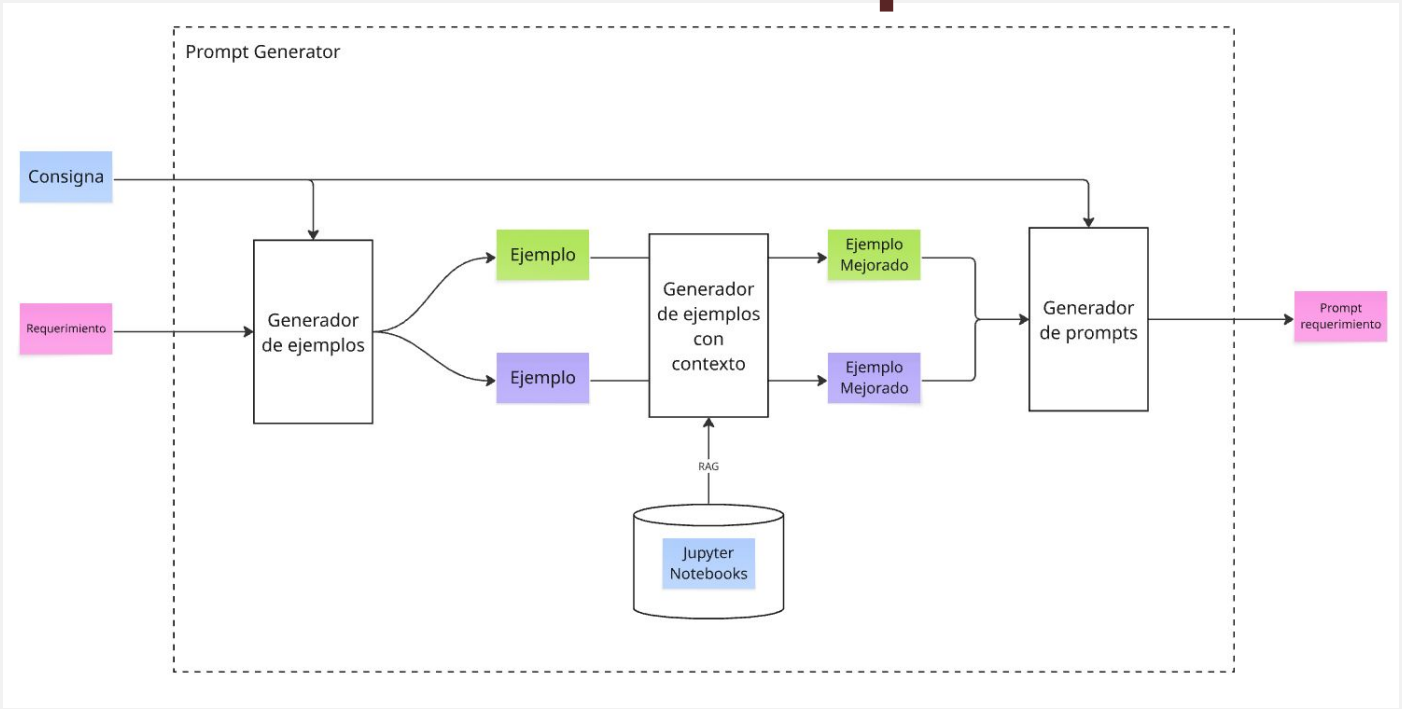
Good example 1:

```
import random
productos = ['producto1', 'producto2', 'producto3']
producto_vendido = random.choice(productos)
unidades_vendidas = random.randint(1, 5)
```

Good example 2:

```
import random
inventario = ['itemA', 'itemB', 'itemC', 'itemD']
producto = random.choice(inventario)
cantidad = random.randint(1, 5)
```

# Generador de Prompts con RAG



# Dataset

11 Jupyter notebooks de la materia Informática General

- Contienen tanto código Python como teoría de la materia en formato Markdown
- En general hay muchas explicaciones seguidas de ejemplos relacionados
- También hay cuestiones más abstractas como buenas prácticas

Como bonus, se hizo un proceso similar para el libro Learn You a Haskell (también en formato de notebooks).

# Decisiones de implementación

- Modelos utilizados: gpt-oss:20b (Ollama) y gpt-4o-mini (OpenAI)
- Vector database: Weaviate
- Modelo de embedding: all-MiniLM-L6-v2
- Plataforma de observabilidad: LangSmith

# Desafíos encontrados

- Diferencias de formato entre los notebooks de Python y los de Haskell
- Estrategia de splitting de Jupyter notebooks (chunk size, separators, celdas vs código)
- Uso de la teoría como apoyo, sin depender exclusivamente de ella
- Limitaciones en prompts autogenerados por Ragas

# Preprocesamiento

- Se convierten las celdas de Jupyter (tipo markdown o código) a un único bloque markdown
- Markdown provee dos sintaxis distintas para títulos, que deben ser uniformizadas en los notebooks para hacer splitting

```
## Heading level 2
```

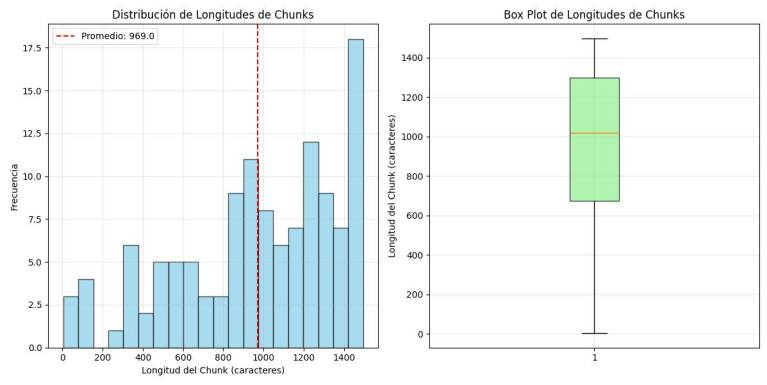
```
Heading level 2
---------------
```

- Además, links markdown y etiquetas HTML no son relevantes para la tarea, por lo que son eliminados

# Text Splitting

- El siguiente paso es el splitting de los notebooks
- Se utilizó un RecursiveCharacterTextSplitter
  CHUNK_SIZE = 1500, CHUNK_OVERLAP = 250
- Para esto se realizaron comparativas de la distribución de longitudes de los chunks

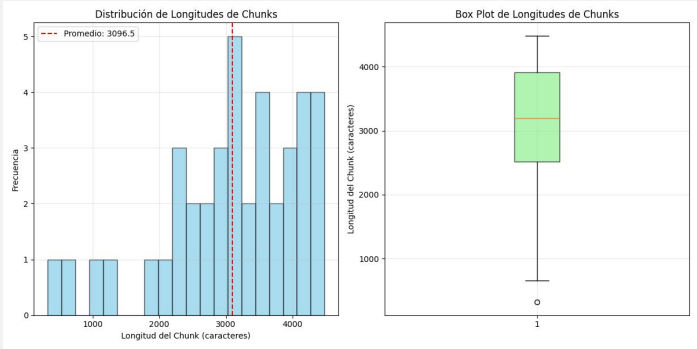
# Text Splitting




CHUNK_SIZE = 1500

CHUNK_OVERLAP = 250

124 chunks




CHUNK_SIZE = 4500

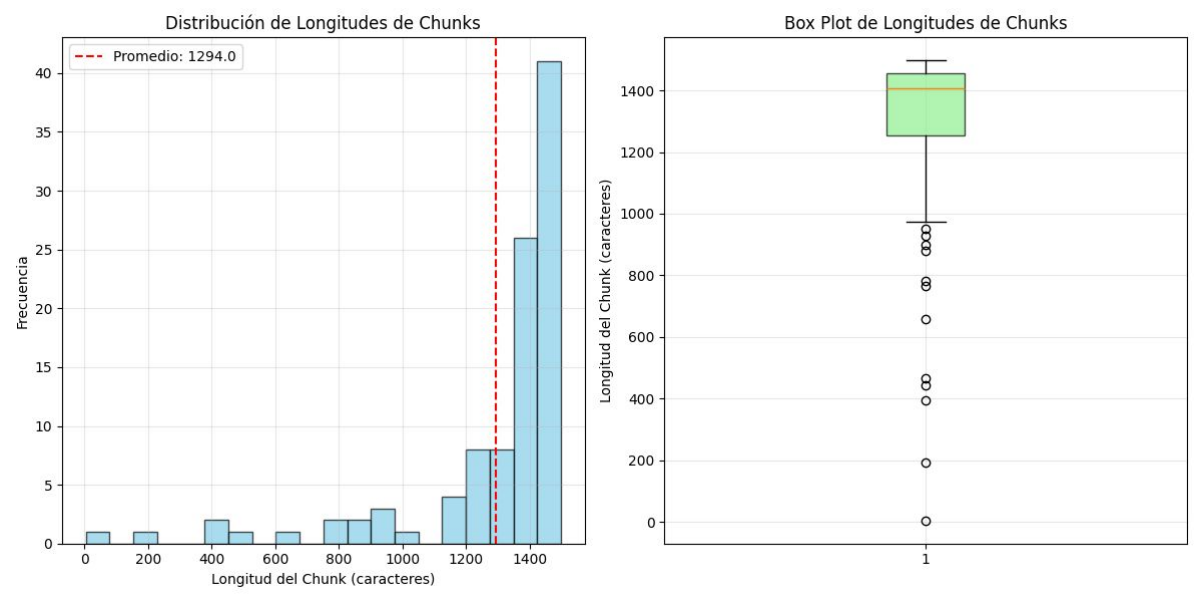
CHUNK_OVERLAP = 750

40 chunks

separators=["\n## ", "\n### ", "```python", "```", "\n\n", "\n", ". ", " ", ""]

Notar como el “\n# ” se descartó por provocar ambigüedades con los comentarios de Python

# Analisis de separators




separators=["\n\n", "\n", ". ", " ", ""]

101 chunks

# Analisis de separators

```
# Listas

Las listas son un tipo de datos que me permite almacenar muchos datos de una forma *secuencial* y
*ordenada*. Se pueden recorrer usando ciclos `for` y 'while'. Con el while las podemos recorrer por
medio de **índices** posiciones, y con el for igual que como lo hicimos con las cadenas.

Veamos un ejemplo de cómo se puede definir una lista:

lista_vacia = []                    ## Los corchetes definen una
                                    # lista, en este caso vacía
lista_con_valores = [ 1 , 2, 3, 4, 5 ]      # Lista creada con valores
                                    # en este cado una cadena
listas_mixtas = [ 1, None, 'cadena', True ]    # Puedes ser distintos tipos
                                    # de datos

print('lista_vacia',lista_vacia)
print('lista_con_valores',lista_con_valores)
print('listas_mixtas',listas_mixtas)
print('listas como literales', [3,2,1]  )      # Puedo definir literales como
                                    # listas
```

```
## Slicing con listas

Para poder acceder a los valores de las listas, puedo usar los corchetes y el índice del valor que quiero
acceder.
De la misma manera que con las cadenas, puedo usar slicing para acceder a algunos elementos de la
lista, tal como se ve en el siguiente ejemplo.
```

Sin los separators semánticos

```
## Slicing con listas

Para poder acceder a los valores de las listas, puedo usar los corchetes y el índice del valor que quiero
acceder.
De la misma manera que con las cadenas, puedo usar slicing para acceder a algunos elementos de la
lista, tal como se ve en el siguiente ejemplo.

lista = [0,1,2,3,4]              # Genero una lista
print('lista',lista)             # Sin corchetes, imprimo toda la lista
print('lista[3]',lista[3])         # un sólo un índice, accedo ese elemento
print('lista[0:2]',lista[0:2])       # rango de elementos
print('lista[::-1]',lista[::-1])     # la lista al revés
```

Con los separators semánticos

Hay una mejor asociación de explicación + código

# Retrieval y Query Expansion

- Se toma el requerimiento y se le inyectan tres ejemplos generados con un LLM
- Con esto, se realiza una búsqueda vectorial en la BD Weaviate que devuelve los fragmentos de teoría más relevantes
- Se arma el contexto teórico a partir de los fragmentos recuperados
- Se mejoran los tres ejemplos inyectados: para cada ejemplo se llama al LLM para alinear el código a la teoría

# Weaviate

- Open source
- Ejecución local
- Escalamiento horizontal
- Definición de esquema explícito

# Esquema de los documentos

```
[{
    "name": "content",
    "dataType": ["text"],
    "description": "The content of the notebook cell",
},
{
    "name": "filename",
    "dataType": ["string"],
    "description": "The name of the notebook file",
},
{
    "name": "notebook_path",
    "dataType": ["string"],
    "description": "The full path to the notebook file",
},
{
    "name": "class_number",
    "dataType": ["int"],
    "description": "The class number extracted from the notebook filename",
},
{
    "name": "dataset",
    "dataType": ["string"],
    "description": "The dataset type (python or haskell)",
}]
```

# Reranking

- Modelo: cross-encoder/ms-marco-MiniLM-L-6-v2
- INITIAL_RETRIEVAL_COUNT = 20, FINAL_RETRIEVAL_COUNT = 5

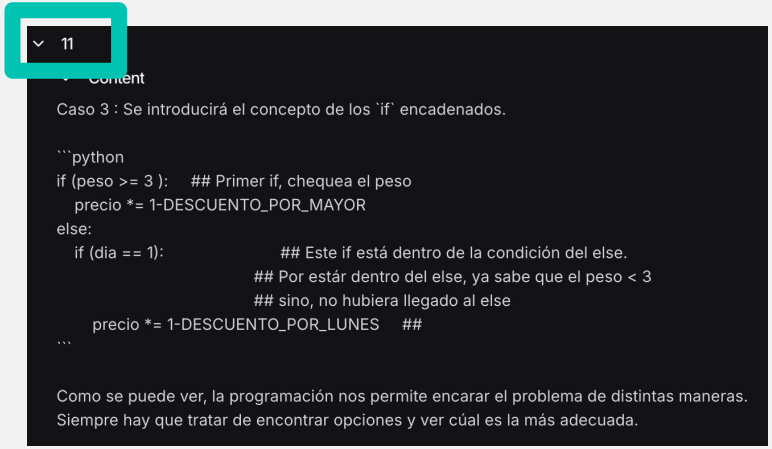
Veamos cómo funciona con el siguiente ejemplo:

**Rubric Requirement:**

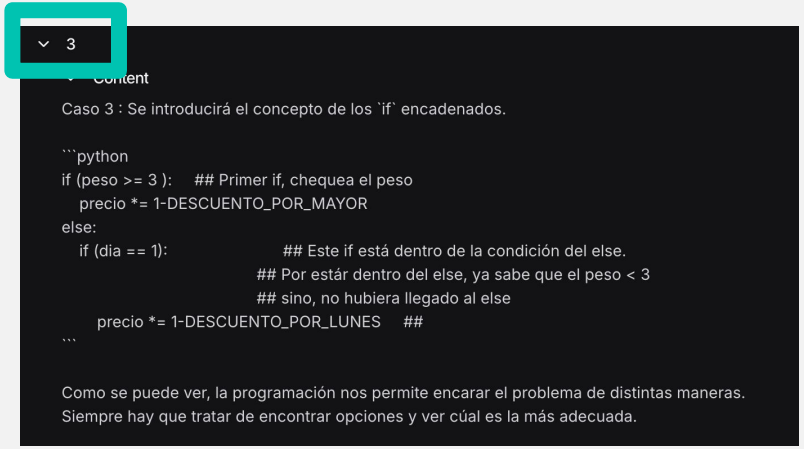
```
Hacer un programa que calcule el precio de una compra de zanahorias.
El programa debe pedir al usuario que ingrese el peso en kilos, el día de la semana y su nombre.
El precio base es de $90 por kilo.
Si el cliente compra 4 kg o más, obtiene un 20% de descuento.
Si el nombre del cliente empieza con la letra “A”, recibe un 10% de descuento especial.
Los descuentos no son acumulables: solo se aplica el mayor de los dos.
```

# Reranking

## Before reranking




## After reranking



# Resultados

Rubric Requirement:

```
Hacer un programa que calcule el precio de una compra de zanahorias.
El programa debe pedir al usuario que ingrese el peso en kilos, el día de la semana y su nombre.
El precio base es de $90 por kilo.
Si el cliente compra 4 kg o más, obtiene un 20% de descuento.
Si el nombre del cliente empieza con la letra “A”, recibe un 10% de descuento especial.
Los descuentos no son acumulables: solo se aplica el mayor de los dos.
```

Ground Truth (Optional):

```
PRECIO_ZANAHORIA_POR_KG = 90       ## En pesos. Variable constante
DESCUENTO_POR_MAYOR = 0.2          ## 20%
DESCUENTO_POR_NOMBRE_A = 0.1       ## 10%

peso = float(input('Ingrese el peso de zanahorias comprado (en kg): '))
nombre = input('Ingrese su nombre: ')

precio = peso * PRECIO_ZANAHORIA_POR_KG

if (peso >= 4 and not nombre.upper().startswith('A')):    ## 4 kg o más, sin descuento por nombre
    precio *= 1 - DESCUENTO_POR_MAYOR
elif (nombre.upper().startswith('A')):                    ## Nombre empieza con A
    precio *= 1 - DESCUENTO_POR_NOMBRE_A
## No hay else, porque si no se cumplen las condiciones no se aplica descuento

print('Compró', peso, 'kg. El precio a pagar es: $', precio)
```

# Resultados

**Original Code:**

```
peso = float(input("Ingrese el peso en kilos: "))
nombre = input("Ingrese su nombre: ")
precio_base = 90
descuento = 0

if peso >= 4:
    descuento = 0.20

if nombre.startswith("A"):
    descuento = max(descuento, 0.10)

precio_final = precio_base * peso * (1 - descuento)
print(f"El precio total de su compra es: ${precio_final:.2f}")
```

Ejemplo original

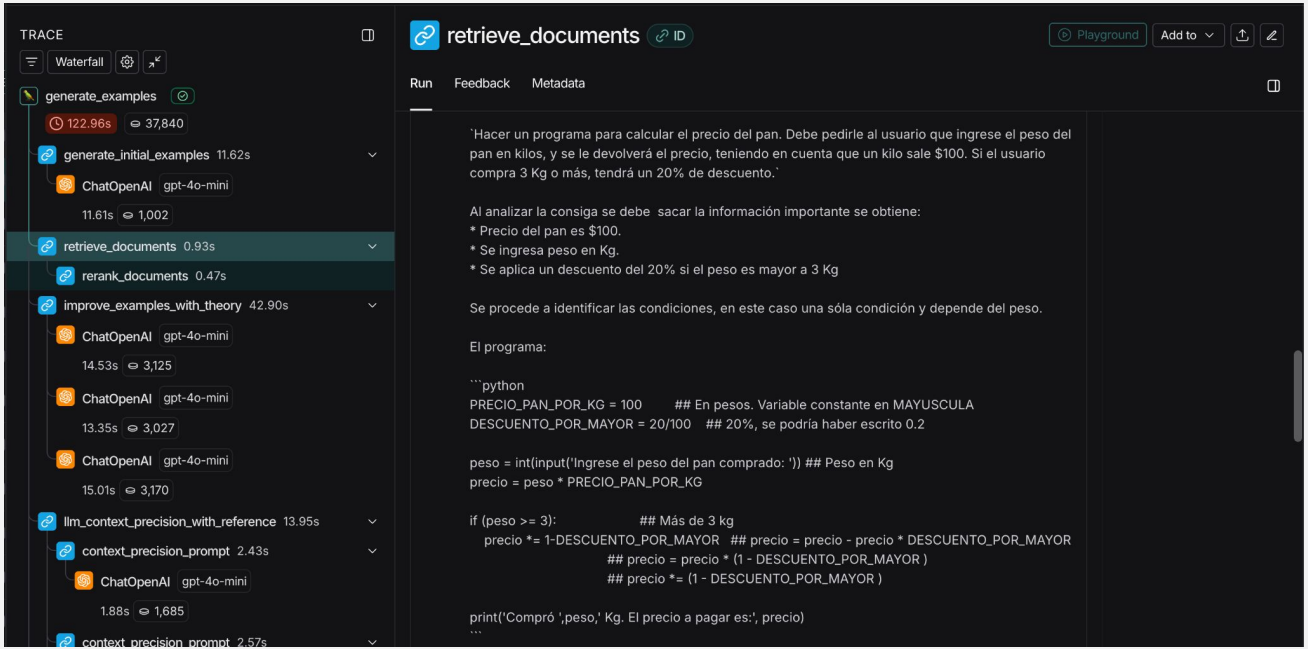
```
PRECIO_ZANAHORIAS_POR_KG = 90         ## En pesos. Variable constante en MAYUSCULA
DESCUENTO_POR_MAYOR = 0.20            ## 20%
DESCUENTO_POR_NOMBRE_A = 0.10          ## 10%

peso = float(input("Ingrese el peso en kilos: ")) ## Peso en Kg
nombre = input("Ingrese su nombre: ")            ## Nombre del cliente
precio_base = peso * PRECIO_ZANAHORIAS_POR_KG    ## Cálculo del precio base

descuento = 0                                    ## Inicializamos descuento

if peso >= 4:                                    ## Verificamos si el peso es igual o mayor a 4 kg
    descuento = DESCUENTO_POR_MAYOR              ## Asignamos el descuento del 20%

if nombre.startswith("A"):                        ## Verificamos si el nombre comienza con "A"
    descuento = max(descuento, DESCUENTO_POR_NOMBRE_A) ## Asignamos el 10% si es mayor

precio_final = precio_base * (1 - descuento)     ## Calculamos el precio final
print(f"El precio total de su compra es: ${precio_final:.2f}")  ## Imprimimos el precio final
```

Ejemplo mejorado

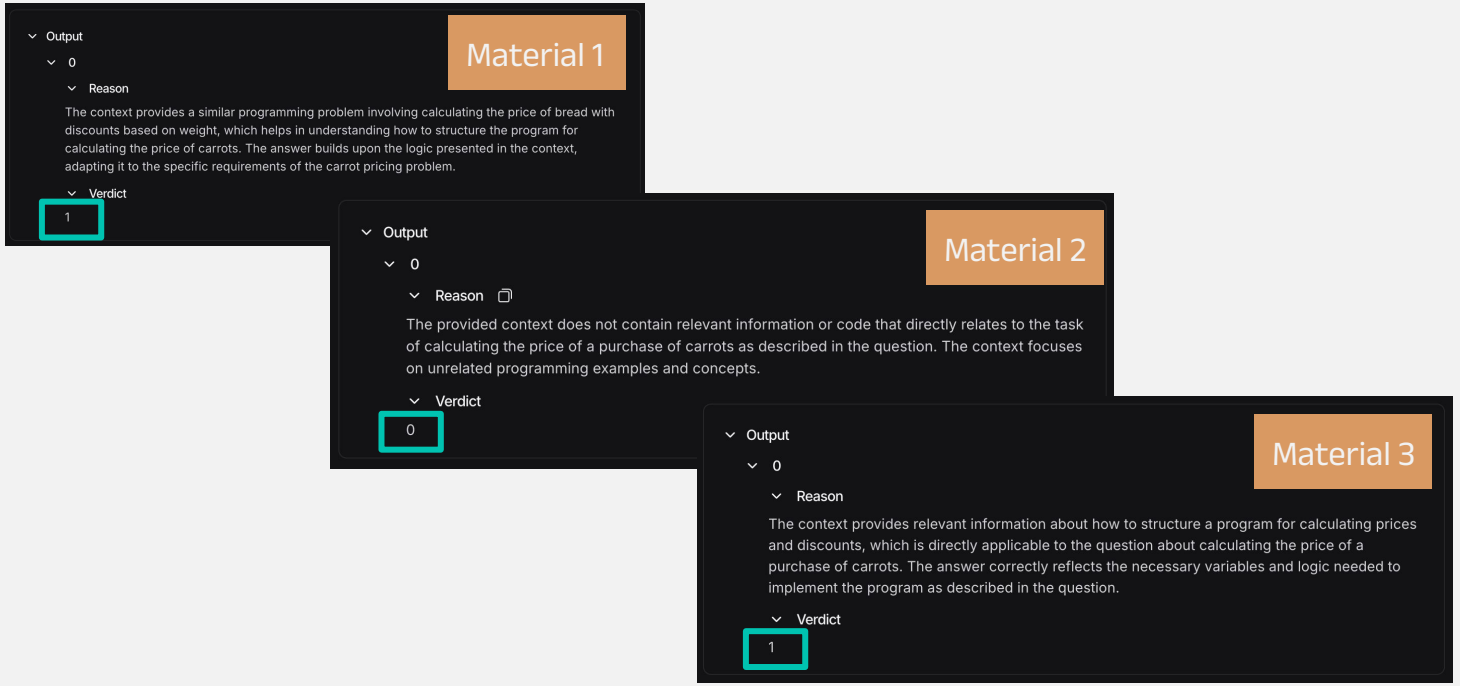
🎯 Context Precision Score (Ragas) 88.7%

🎯 Answer Relevancy Score (Custom) 84.2%

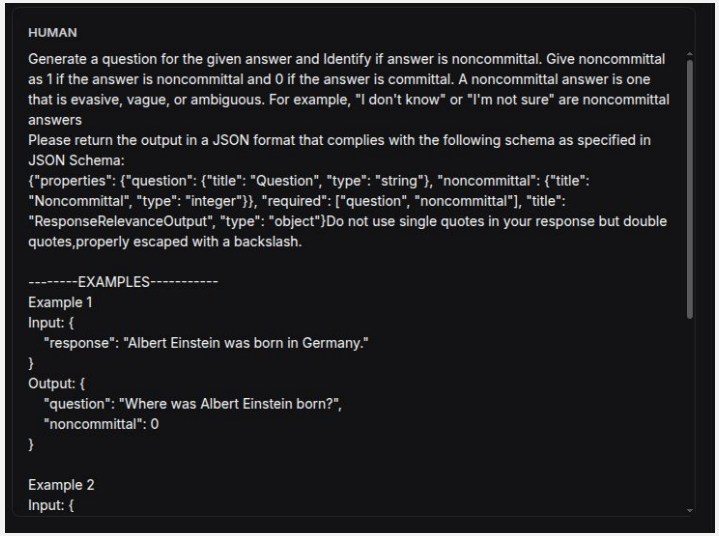
# Resultados

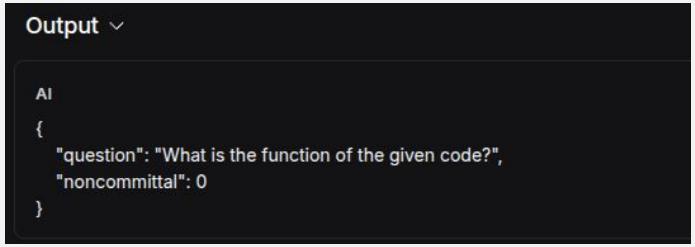

TRACE
Waterfall
generate_examples
122.96s
37,840
generate_initial_examples 11.62s
ChatOpenAI gpt-4o-mini
11.61s
1,002
retrieve_documents 0.93s
rerank_documents 0.47s
improve_examples_with_theory 42.90s
ChatOpenAI gpt-4o-mini
14.53s
3,125
ChatOpenAI gpt-4o-mini
13.35s
3,027
ChatOpenAI gpt-4o-mini
15.01s
3,170
llm_context_precision_with_reference 13.95s
context_precision_prompt 2.43s
ChatOpenAI gpt-4o-mini
1.88s
1,685
context_precision_prompt 2.57s
retrieve_documents
ID
Playground
Add to
Run
Feedback
Metadata
`Hacer un programa para calcular el precio del pan. Debe pedirle al usuario que ingrese el peso del pan en kilos, y se le devolverá el precio, teniendo en cuenta que un kilo sale $100. Si el usuario compra 3 Kg o más, tendrá un 20% de descuento.`
Al analizar la consiga se debe sacar la información importante se obtiene:
* Precio del pan es $100.
* Se ingresa peso en Kg.
* Se aplica un descuento del 20% si el peso es mayor a 3 Kg
Se procede a identificar las condiciones, en este caso una sóla condición y depende del peso.
El programa:
```python
PRECIO_PAN_POR_KG = 100 ## En pesos. Variable constante en MAYUSCULA
DESCUENTO_POR_MAYOR = 20/100 ## 20%, se podría haber escrito 0.2
peso = int(input('Ingrese el peso del pan comprado: ')) ## Peso en Kg
precio = peso * PRECIO_PAN_POR_KG
if (peso >= 3): ## Más de 3 kg
precio *= 1-DESCUENTO_POR_MAYOR ## precio = precio - precio * DESCUENTO_POR_MAYOR
## precio = precio * (1 - DESCUENTO_POR_MAYOR )
## precio *= (1 - DESCUENTO_POR_MAYOR )
print('Compró ',peso,' Kg. El precio a pagar es:', precio)
```

# Métricas: Context Precision



# Métricas: Answer Relevancy




Ragas pide que se genere una pregunta a partir del código, lo cual es incompatible con el proyecto

# Métricas: Answer Relevancy (custom)

HUMAN

Eres un instructor de programación en Python. Dado el siguiente código Python, infiere 3 posibles requerimientos o consignas diferentes que podrían haber llevado a un estudiante a escribir este código.

EJEMPLO DE REFERENCIA:

Código de ejemplo:

python

```python
def sumar_pares(lista):
    suma = 0
    for numero in lista:
        if numero % 2 == 0:
            suma += numero
    return suma
```

Rubricas generadas:
<rubric>
<requirement>La función debe tomar una lista de números y devolver la suma de todos los números pares</requirement>
</rubric>

<rubric>
<requirement>`sumar_pares` debe recorrer una lista y sumar únicamente los elementos que son divisibles por 2</requirement>
</rubric>

<rubric>
<requirement>Debe calcular la suma total de los números pares presentes en una lista de enteros</requirement>
</rubric>

---

# Métricas: Answer Relevancy (custom)

Output

AI

```xml
<rubric>
<requirement>La aplicación debe calcular el precio total de zanahorias según el peso ingresado por el cliente, aplicando descuentos específicos basados en el peso y el nombre del cliente.</requirement>
</rubric>

<rubric>
<requirement>El programa debe permitir al usuario ingresar el peso de las zanahorias y su nombre, y luego determinar el precio a pagar, considerando un descuento del 20% si el peso es igual o mayor a 4 kg y un descuento adicional del 10% si el nombre comienza con la letra "A".</requirement>
</rubric>

<rubric>
<requirement>Se debe implementar un sistema que calcule el costo final de zanahorias, aplicando un descuento del 20% para compras de 4 kg o más, y un descuento adicional del 10% para clientes cuyo nombre inicie con "A", mostrando el precio total al finalizar.</requirement>
</rubric>
```

# Métricas: Answer Relevancy (custom)

Dados los tres requirements generados a partir del prompt provisto, se calculan sus embeddings, y con ellos, el Answer Relevancy a partir de la **similitud coseno**

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \text{cosine similarity}(E_{g_i}, E_o)$$

# Resultados

**Rubric Requirement:**

```
Para cada venta, se debe elegir al azar un producto del inventario actual y la cantidad vendida (entre 1 y
5)
```

**Ground Truth (Optional):**

```
import random
productos = ["Producto1","Producto2","Producto3"]
producto_vendido = productos[random.randint(0,len(productos)-1)]
cantidad_vendida = random.randint(1,5)
```

Provide a reference answer to evaluate context precision using Ragas. This will show how well the retrieved contexts align with your expected answer.

# Resultados

**Original Code:**

```
productos = ["manzana", "banana", "naranja", "pera", "uva"]
import random

producto_vendido = productos[random.randrange(len(productos))]
cantidad_vendida = random.randint(1, 5)

print("Producto vendido:", producto_vendido)
print("Cantidad vendida:", cantidad_vendida)
```

Ejemplo original

```
import random

productos = ["manzana", "banana", "naranja", "pera", "uva"]

# Seleccionamos un índice aleatorio para elegir un producto
indice_producto = random.randint(0, len(productos) - 1)
producto_vendido = productos[indice_producto]

# Generamos una cantidad aleatoria entre 1 y 5
cantidad_vendida = random.randint(1, 5)

# Imprimimos el resultado
print("Producto vendido:", producto_vendido)
print("Cantidad vendida:", cantidad_vendida)
```

Ejemplo mejorado

# Resultados

**Rubric Requirement:**

Para cada venta, se debe elegir al azar un producto del inventario actual y la cantidad vendida (entre 1 y 5)

🎯 **Context Precision Score (Ragas)** 100.0%

🎯 **Answer Relevancy Score (Custom)** 59.4%

🔍 **Generated Rubrics (3):**

1. El programa debe seleccionar aleatoriamente un producto de una lista predefinida y mostrarlo como el producto vendido.
2. Se requiere que el código genere una cantidad aleatoria entre 1 y 5 que represente la cantidad de un producto vendido y la imprima junto al nombre del producto.
3. La funcionalidad debe permitir que el sistema elija un producto de manera aleatoria y asigne una cantidad de venta aleatoria, mostrando ambos resultados en la salida.

# ¡Gracias!