

**Proyecto Fin de Máster**

Máster en Big Data, Inteligencia Artificial y Data Science

**HealthBite:**

Aplicación con Visión Artificial y Lenguaje Natural para la nutrición personalizada a partir de ingredientes y estado del usuario

Septiembre 2025

**Autor:** Óscar Xu Zhou

Guión:

* Resumen
* 1) Introducción
  + 1.1 Contexto y justificación del proyecto
  + 1.2 Objetivos del proyecto
* 2) Flujo de la aplicación y algoritmos utilizados
* 3) Análisis exploratorio de datos (EDA)
* 4) Creación del modelo YOLO
  + 5.1) Obtención de datasets
  + 5.2) Entrenamiento del modelo
  + 5.3) Evaluación del modelo y ejemplos de predicción
* 5) Creación de modelo NLP
  + 6.1) Obtención de datasets
  + 6.2) Entrenamiento del modelo
  + 6.3) Evaluación del modelo y ejemplos de predicción
* 6) Algoritmo de recomendación y RAG LLM como juez
  + 7.1) Combinación de los modelos YOLO + NLP
  + 7.2) Traducción del dataset utilizado de recetas
  + 7.3) Algoritmo de recomendación por reglas y puntuación
  + 7.4) RAG LLM como juez final y prompt engineering
* Productivización del sistema
* 7) Conclusiones y resultados
  + 8.1) Limitaciones actuales
  + 8.2) Trabajo futuro

Contents

[**1. Introducción** 4](#_Toc208514912)

[1.1 Contexto y justificación del proyecto 4](#_Toc208514913)

[1.2 Objetivos del proyecto 4](#_Toc208514914)

[**Flujo de la aplicación y algoritmos utilizados** 5](#_Toc208514915)

[**Análisis exploratorio de datos (EDA)** 6](#_Toc208514916)

[**Creación del modelo YOLO** 8](#_Toc208514917)

[Obtención de datasets 9](#_Toc208514918)

[Entrenamiento del modelo 10](#_Toc208514919)

[Evaluación del modelo y ejemplos de predicción 11](#_Toc208514920)

[Ejemplo de uso del modelo final 13](#_Toc208514921)

[**Construcción del modelo NLP** 14](#_Toc208514922)

[Obtención de datasets 14](#_Toc208514923)

[Entrenamiento del modelo 15](#_Toc208514924)

[Evaluación del modelo y ejemplos de predicción 16](#_Toc208514925)

[En conclusión, para un primer prototipo se selecciona el modelo con partición aleatoria como modelo final, ya que presenta una pérdida inferior, aunque con métricas ligeramente más bajas en accuracy y recall. Se destaca, no obstante, la necesidad de ampliar y actualizar el dataset con descripciones reales de usuarios a medida que la aplicación se utilice y puede coleccionas más datos, con el fin de reforzar la robustez y validez del modelo en escenarios más diversos. 18](#_Toc208514926)

[Ejemplo de uso del modelo final 18](#_Toc208514927)

# **1. Introducción**

## 1.1 Contexto y justificación del proyecto

El desperdicio alimenticio es uno de los grandes problemas que afronta la sociedad actual. Se estima que en la Unión Europea los hogares generan un 54% del desperdicio total, lo que equivale a aproximadamente 72 kilos de desperdicio por persona. (Eurostat, 2024). Este fenómeno no solo implica pérdidas económicas, sino también un fuerte impacto ambiental, dado que se desaprovecha toda la energía invertida en la producción y la descomposición de los alimentos en vertederos genera gases de efecto invernadero.

Diversos estudios señalan que el uso cotidiano de la nevera está relacionado con desperdicio alimenticio. Un estudio sociológico descubrió que, en muchos casos, los productos quedan olvidados o quedan ocultos por la posición, causando que caduquen sin ser consumidos (Heidenstrøm & Hebrok, 2020) Por otra parte, otro estudio reveló que el uso de la nevera es asociado a un aumento del 24.35% de desperdicio alimenticio (Longqiang Zhao, Min, Wang, & Yu, 2024). Estos datos evidencian la necesidad de soluciones que ayuden a la optimización del consumo de los alimentos almacenados en la nevera.

Adicionalmente al contexto de desperdicio, la nutrición personalizada se ha consolidado como un campo emergente alcanzando grandes mejoras sobre todo con los últimos avances en inteligencia artificial. Aunque existen aplicaciones móviles que ofrecen recomendaciones dietéticas, una gran mayoría se han enfocado a proporcionar información tras un escaneo códigos de barras o proponer recetas aleatorias o genéricas, mayoritariamente orientadas a dietas saludables o fitness. Estas soluciones rara vez tienen en cuenta los ingredientes realmente disponibles en el hogar ni cubren las necesidades dietéticas específicas del usuario.

Este proyecto propone el desarrollo de HealthBite, una aplicación inteligente basada en visión artificial, algoritmos de lenguaje natural y modelos de lenguaje de gran escala (LLM). Su objetivo es recomendar recetas que, por un lado, prioricen el uso de los ingredientes disponibles en la nevera para reducir el desperdicio alimenticio, y por otro, contribuyan a mejorar la salud dietética del usuario al cubrir carencias nutricionales potenciales inferidas a partir de descripciones de estado anímico/físico. De este modo, la propuesta combina sostenibilidad, innovación tecnológica y bienestar personal en una única solución escalable.

## 1.2 Objetivos del proyecto

**Objetivo principal:**

Desarrollar una aplicación inteligente capaz de recomendar recetas considerando tanto los ingredientes disponibles en la nevera del usuario como sus posibles deficiencias nutricionales, inferidas a partir de descripciones de su estado anímico o físico. Adicionalmente, será capaz de explicar las razones de cada recomendación.

**Objetivos específicos:**

* Realizar un análisis exploratorio de datos (EDA) sobre resultados de encuestas de consumición alimentario en hogares de España, con el fin de obtener mejor entendimiento sobre patrones y productos más consumidos
* Creación de un modelo de visión artificial (YOLO) para identificar automáticamente los ingredientes disponibles en la nevera a partir de imágenes
* Entrenar un modelo de procesamiento del lenguaje natural (NLP) capaz de clasificar descripciones del estado anímico/físico del usuario en síntomas y asociarlas con posibles deficiencias nutricionales
* Implementar un sistema de traducción automática a gran escala mediante modelos NLP para adaptar el dataset de recetas del inglés al español
* Construir un sistema de puntuación que permita ordenar las recetas en función de su afinidad al contexto específico del usuario (ingredientes disponibles y necesidades nutricionales).
* Desarrollar e integrar un LLM con RAG, ajustado mediante técnicas de prompt engineering, que actúe como juez final para validar que las condiciones de recomendación se cumplen y genere una explicación clara para el usuario.
* Integrar todos los modelos en una aplicación interactiva con Streamlit, lista para ser utilizada en un entorno real por usuarios finales.

# A diagram of a diagram AI-generated content may be incorrect.**2. Flujo de la aplicación y algoritmos utilizados**

*Imagen 1: Diagrama de flujo de la estructura y sistema de la aplicación*

En esta sección detallaremos el flujo completo del funcionamiento de la aplicación a construir. Tal y como se puede observar en el diagrama de flujo, el proceso hasta llegar a la recomendación personalizada final de recetas pasa por varias fases:

**Fase de inputs:**

En esta fase se obtienen los inputs necesarios para los modelos a utilizar posteriormente:

* Se recogen dos inputs del usuario: una foto de su nevera y una descripción breve de su estado físico/anímico.
* Se carga internamente un dataset de recetas (2+ millones de registros) que servirá como base para las recomendaciones. **Fase de entrenamiento y aplicación de los modelos:**

**Fase de entrenamiento y aplicación de modelos**

En este paso se entrenarán y se aplicarán tres modelos principales:

* **Detección de objetos *(YOLO, You Only Look Once):*** es un modelo de detección de objetos basado en redes convolucionales que predice clases de objetos a partir de cajas delimitadas. Se ha fine-tuneado con fotos curadas de ingredientes en la nevera combinando datos públicos con fotos sintéticas generadas por scripts de Python y data augmentation.
* **NLP para síntomas (base BERT):** se utiliza un modelo *Transformer* capaz de entender descripciones en español y clasificar a partir de del texto de estado síntomas y posibles deficiencias nutricionales. Se ha fine-tuneado con descripciones anímico-físcas con un dataset sintético compilado a partir de fuentes médicas y clínicas
* Por último, también se utiliza el **modelo Marian MT**, modelo encoder-decoder para la traducción masiva del dataset de recetas de más de 2 millones de filas del inglés al español.

**Outputs de los modelos:**

En esta fase, se recogen las salidas de los diferentes modelos:

* Ingredientes disponibles (a partir de la foto de nevera usando YOLO)
* Síntomas y posibles deficiencias en nutrientes (a partir de la descripción)
* Recetas en español (a partir de la traducción masiva)

Estos resultados alimentarán el sistema de puntuación

**Sistema de puntuación:**

Se ha desarrollado un algoritmo que puntúa y hace un ranking de cada receta en base a una serie de métricas. Se evalúan aspectos como: si la receta requiere ingredientes que ya dispone el usuario en la nevera, si la receta contiene ingredientes que ayudarían con la deficiencia de nutrientes del usuario, etc.

**Recomendación final:**

En esta última fase, se toman las 10 recetas mejor puntuadas y un LLM (Large Language Model), con RAG (Retrieval-Augmented Generation), actúa como juez para seleccionar las tres mejores finales. El modelo recupera evidencia relevante (ingredientes detectados, síntomas/deficiencias y fichas de las recetas) y se guía por instrucciones (prompting) con criterios explícitos. Con ese contexto, reevalúa las candidatas y descarta las que incumplen restricciones o requieren ingredientes críticos ausentes. Devuelve las tres recetas elegidas junto con una explicación breve del razonamiento.

# **Análisis exploratorio de datos (EDA)**

Con el objetivo final de construir una aplicación inteligente capaz de recomendar recetas personalizadas, el primer paso es realizar un análisis exploratorio de datos para entender qué categorías de alimentos tiene mayor consumo en la sociedad española. Como para la aplicación se construirá un modelo YOLO capaz de identificar ingredientes en fotografías de neveras, necesitamos priorizar una cantidad manejable de clases. Dado que las combinaciones de ingredientes pueden ser infinitas, esto puede suponer costes computacionales muy elevados a la hora del entrenamiento del modelo. Por ello, se he decidido fijar un tope de 30 clases a entrenar y predecir para equilibrar cobertura y coste computacional. Para no sacrificar calidad el EDA se apoya en resultados de encuestas oficiales para seleccionar los alimentos más consumidos y con mayor crecimiento, de modo que la aplicación aporte valor pese a la escala reducida.

**Obtención de dataset:**

Los datos han sido obtenidos combinando diferentes resultados de encuestas a hogares sobre consumo de alimentos publicados periódicamente (en archivo Excel, anual o semestralmente) por el Ministerio de Agricultura, Pesca y Alimentación (MAPA). Se ha construido un script de Python que recorre cada archivo preprocesando y estandarizando formatos y esquemas y combina todos los archivos en un único DataFrame (para más información, véase el anexo). El dataset incluye métricas como consumo per cápita, gasto per cápita y penetración de categoría (porcentaje de hogares consumidores).

**Depuración y procesamiento realizado:**

Además de estandarizar, se ha realizado una depuración y filtro del set. Los datos publicados contienen muchas filas que no son representativas o de utilidad. Se ha decidido codificar cada fila del dataset para poder filtrar después y obtener solamente filas que son de interés.

En primer lugar, se codifica cada fila basándonos en el nombre original en categorías diferentes del 0 al 3. Solamente nos quedaremos con la categoría 1. A continuación se muestra la lógica de codificación:

* Categoría 0: el nombre es demasiado genérico (ej.: “total alimentación”, “total carnes”).
* Categoría 1: el nombre es representativo (ej.: “Huevos”, “Miel”).
* Categoría 2: el nombre es demasiado específico y puede estar englobado en otra fila (ej.: “Huevos a granel”, “Huevos envasados”).
* Categoría 3: nombres ambiguos que no aportan valor (ej.: “Otras carnes”, “Otras aves”).

En segundo lugar, se refinan algunos nombres para mejor legibilidad (ej: T.Huevos UNDS -> “Huevos”). Por último, dado que hay demasiados nombres únicos, realizar un análisis exploratorio sería difícil. Se ha decidido agrupar diferentes alimentos bajo categorías de la pirámide alimenticia.

Se muestra dos ejemplos de cómo quedarían las filas del dataset:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre original | Categoría codificada | Nombre refinado | Categoría de pirámide alimenticia |
| T.Huevos Unds | 1 | Huevos | Lácteos, Derivados y Huevos |
| Cons.pescado atún | 1 | Conserva atún | Conservas y precocinados |

**Análisis de datos:**

El análisis exploratorio se centra en identificar productos con mayor penetración en hogares, mayor consumo per cápita y mayor crecimiento porcentual (tanto en penetración como en consumo). Con ello cubrimos los alimentos históricamente populares y, a la vez, los que muestran adopción acelerada. A modo ilustrativo se presentan algunas gráficas (para el conjunto completo, véase el anexo, sección XXX).

A graph of different colored bars

AI-generated content may be incorrect.

A graph showing a line

AI-generated content may be incorrect.

*\*El axis y representa el crecimiento porcentual, que para algunos productos es considerablemente alto*

Tras analizar las gráficas y los números, se han utilizado los siguientes criterios para decidir qué clases escoger e integrar dentro del proyecto para este primer prototipo:

* **Criterio de mayor métrica:** Priorizar productos con valores altos en las métricas observadas (penetración, consumo per cápita, crecimiento).
* **Criterio contextual:** Se debe tener en cuenta que las clases serán las que podrá predecir nuestro futuro modelo de detección de objetos en la nevera. Hay ciertos alimentos que, a pesar de tener buenas métricas, probablemente no se almacenarían en la nevera (e.j: conservas no abiertas de lata, pasta, etc.). Estos alimentos se descartan.
* **Criterio transfer-learning:** Favorecer clases con presencia en datasets públicos para aprovechar mejor el aprendizaje por transferencia.

A continuación, se ofrece un resumen de los alimentos seleccionados, en total 30 clases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Verduras y hortalizas** | **Pescados y mariscos** | **Lácteos, derivados y huevos** | **Carnes** | **Frutas** |
| Tomate  Cebolla  Patata  Lechuga/Endivia  Zanahorias  Calabacines  Pepino  Champiñones  Brocoli  Coliflor | Merluza  Gambas/Langostinos  Mix marisco/molusco  Lubina  Salmón | Leche  Huevos  Yogur  Queso  Mantequilla | Carne pollo  Carne cerdo  Carne vacuno  Salchichas  Carne pavo | Plátano  Aguacate  Sandía  Limón  Manzanas  Aguacate |

Una vez seleccionadas las clases, se han analizado los resultados por grupos socioeconómicos para ver si hay demasiado desbalance en métricas en las clases de productos escogidas a predecir posteriormente en el modelo YOLO. Se han comparado específicamente los porcentajes de penetración en el hogar por poder económico y por la edad de los participantes de la encuesta. Podemos observar en las siguientes gráficas las siguientes conclusiones:

* Todos los productos seleccionados tienen mayor % de penetración en hogares con nivel alto y menor porcentaje en hogares de nivel bajo. Sin embargo, en general no hay demasiada disparidad
* En cuanto a edad, observamos que el porcentaje de penetración también está bastante distribuido por todos los grupos de edad, a excepción de los productos de pescado, que alcanzan solamente un 30% de particulares de menos de 30 años.

En general, las clases escogidas de las diferentes categorías tienen alto porcentaje de consumo y no hay demasiado desbalance entre grupos sociodemográficos.

A graph showing different colored bars

AI-generated content may be incorrect.

A graph showing different colored bars

AI-generated content may be incorrect.

# **Creación del modelo YOLO**

En esta sección se describe la construcción y ajuste de un detector de objetos YOLO para identificar ingredientes en fotografías de neveras. YOLO es un modelo de visión por computador basado en redes neuronales convolucionales que, en una única pasada por la imagen, predice simultáneamente cajas delimitadoras, confianza y clase de cada objeto, permitiendo detección en tiempo real. (Redmon, 2015)

la construcción del set de datos utilizado posteriormente para entrenar el modelo de detección de objetos YOLO (You Only Look Once), un algoritmo de visión artificial que permite localizar e identificar objetos mediante cajas delimitadas (bounding boxes en inglés).

## Obtención de datasets

El primer paso para construir el modelo YOLO definitivo ha sido obtener el set de datos. El set de datos definitivo se construirá a partir de una combinación de:

* **Datasets públicos:** Se han conseguido varios datasets con fotografías de ingredientes en la nevera, principalmente descargados de la plataforma *Roboflow (plataforma simlar a Kaggle)*. Los diferentes datasets públicos han sido combinados en uno definitivo, no obstante, previamente se ha estandariza el documento data.yaml donde cada clase se mapea a su id correspondiente para la consistencia.

A pesar de haber datasets públicos, no se disponen de fotografías de todas las 30 clases de alimentos seleccionadas previamente en el análisis exploratorio. Adicionalmente, algunas clases, a pesar de aparecer en algunas fotografías, están poco representadas.

* **Datasets sintéticos:** dado que no existen fotografías de todas las clases necesarias, se ha decidido generar a partir de un script de Python fotos sintéticas de ingredientes en las neveras. El script toma imágenes individuales de ingredientes, les elimina el fondo y las coloca aleatoriamente sobre distintos fondos de neveras vacías para simular fotos realistas que podría haber tomado un usuario. Esta técnica nos permite suplementar los datasets públicos cuyas fotos no incluyen algunas de las clases que necesitamos identificar.

A diagram of a refrigerator

AI-generated content may be incorrect.Se muestra a continuación una representación gráfica del proceso de generación de datos sintéticos.

* En primer lugar, se descargan fotografías de ingredientes que no se han podido encontrar en datasets públicos o que tienen poca representación. Estas fotos son procesadas utilizando la librería *rembg* eliminando los fondos que contengan.
* En segundo lugar, se descargan fotos de neveras vacías
* Una vez listas las fotos de nevera, se delimitan cajas que sirven como coordenadas de áreas donde se le permite al script de Python posicionar ingredientes aleatoriamente respetando sin sobrepasar el grado de solapamiento definido y sin posicionar demasiados ingredientes en la misma área.
* Con todo listo, un script de Python (véase el script en el anexo XXXX) toma diferentes ingredientes sin fondo y los posiciona aleatoriamente en las áreas delimitadas de diferentes fotos de nevera para simular fotos realistas.
* A refrigerator with a few objects falling

  AI-generated content may be incorrect.Por último, para aumentar la diversidad del dataset sintético, se usa la función *augment\_synthetic\_dataset (véase anexo XXX)* que aplica técnicas de aumentación (rotaciones, inversiones, cambios de tono de color, ruido gaussiano, etc.) ya que las imágenes reales tomadas por los usuarios pueden tener ángulos, iluminación, orientaciones diferentes, etc. La técnica de *data augmentation* ayuda a simular situaciones más realistas y también evita que el modelo simplemente memorice patrones.

**A graph of blue bars with black text

AI-generated content may be incorrect.A graph with blue bars

AI-generated content may be incorrect.**Procedemos a comparar gráficamente cómo ha evolucionado el conteo de imágenes en las que aparece cada ingrediente. Podemos observar que antes de la creación del dataset sintético (gráfico izquierdo) había clases con muy poca o nula representación como pepino, limón, sandía, salmón, etc. Tras incorporar el dataset sintético a los datasets públicos, observamos que ahora todas las clases tienen cierta representación. El dataset final contiene 17081 imágenes diferentes.

## Entrenamiento del modelo

Se han entrenado tres modelos diferentes a partir de la versión base YOLOv11n (nano), seleccionada por su bajo coste computacional y eficiencia en entornos con recursos limitados. Este modelo contiene aproximadamente 2.6 millones de parámetros entrenables y requiere alrededor de 6.5 mil millones de operaciones matemáticas (*GFLOPs*) por imagen. El modelo original ha sido preentrenado con el conjunto de datos COCO (*Common Objects in Context*), que incluye más de 330 000 imágenes y alrededor de 1.5 millones de instancias de objetos de uso general. En la configuración estándar, las imágenes de entrada se redimensionan a 640x640 píxeles. (Jocher & Qiu, 2024)

En esta sección, trabajamos en reentrenar el modelo base aplicando transfer learning para poder adaptarlo a nuestro problema y contexto específicos, la detección de diferentes clases de ingredientes en fotografías de nevera.

**Características comunes de los modelos reentrenados**

Los tres modelos mantienen la misma estructura que la versión base, con 181 capas que incluyen convoluciones, normalizaciones y funciones de activación. El conjunto de datos utilizado se ha dividido en un 70% para entrenamiento, 15% para validación y 15% para prueba, contemplando un total de 30 clases de salida. El conjunto de entrenamiento permite el aprendizaje al modelo y la actualización de los parámetros del modelo, el de validación se emplea para monitorizar la función de pérdida y métricas de rendimiento durante el aprendizaje, y finalmente el conjunto de prueba evalúa la capacidad de generalización sobre datos no vistos.

**Modelos entrenados**

* ***Modelo 1 - 5 epochs sin datos sintéticos*:** Se ha entrenado un modelo base sencillo de tan solo 5 epochs (que representan la cantidad de pasadas completas por todo el conjunto de datos de entrenamiento). En este modelo se usan solamente con los datasets públicos descargados sin el sintético generado. La baja representación de ciertas clases limita su rendimiento, pero este modelo sirve como línea base para evaluar la calidad de los datos públicos.
* ***Modelo 2 - 10 epochs con datos sintéticos*:** Incluye un modelo de 10 epochs donde se ha incluido un dataset generado de manera sintética para cubrir ingredientes con baja o nula representación. Este modelo nos ayudará a validar si las métricas y el rendimiento empeoran al introducir fotos artificiales en el entrenamiento para mejorar la cobertura de clases.
* ***Modelo 3 - 100 epochs con datos sintéticos*:** Modelo final (100 epochs), entrenado durante más iteraciones y con una resolución superior a la estándar. Esta configuración busca mejorar la detección de objetos pequeños a costa de un mayor consumo de recursos y menor velocidad de inferencia.

## Evaluación del modelo y ejemplos de predicción

A continuación, se detallan los resultados obtenidos en el entrenamiento de cada modelo y sus métricas:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Métrica | Modelo 1 (5 epochs) | Modelo 2 (10 epochs) | Modelo 3 (100 epochs) | Aceptable | Bueno | SOTA (State of the Art) |
| Precisión | ~0.77 | ~0.90 | ~0.96 | >0.6 | >0.8 | >0.9 |
| Recall | ~0.59 | ~0.79 | ~0.90 | >0.5 | >0.7 | >0.85 |
| mAP50 | ~0.67 | ~0.87 | ~0.95 | 0.5–0.6 | 0.7–0.8 | >0.85 |
| mAP50-95 | ~0.45 | ~0.64 | ~0.83 | 0.3–0.4 | 0.5–0.6 | >0.65 |

A graph of different colored bars

AI-generated content may be incorrect.

Las métricas utilizadas para evaluar los modelos son los siguientes:

* **Precisión:** mide la fiabilidad de las detecciones y corresponde a la proporción de predicciones positivas que han sido correctas. Fórmula:
* **Recall:** mide la capacidad del modelo para encontrar objetos reales presentes en las imágenes. Un valor alto indica que se detecta la mayoría de los objetos. Fórmula:
* **mAP50:** corresponde al *Mean Average Precision* calculado con un umbral de IoU = 0.50. El IoU (*Intersection over Union*) mide el solapamiento entre la caja predicha y la caja real. Con IoU = 0.50 se considera que una predicción es correcta si al menos la mitad de la caja se solapa con el objeto real. Esta métrica es más permisiva.
* **mAP50-95:** se calcula como el promedio del *Average Precision* en diferentes umbrales de IoU: 0.50, 0.55, 0.60, … hasta 0.95. De esta manera se evalúa no solo si el modelo detecta los objetos, sino también como de bien se ajustan las cajas de predicción a los objetos reales. Esta métrica es mucho más estricta.

Para comparación, se establecen umbrales de referencia basados en valores reportados habitualmente en la literatura como *benchmarking*. En el dataset COCO, valores de mAP50-95 en torno a 0.3–0.4 se consideran aceptables para modelos ligeros, entre 0.5–0.6 se consideran competitivos, y superiores a 0.65 suelen asociarse con el estado del arte, según resultados recientes de modelos como YOLOv8, YOLOv11 y DETR.

Se utiliza como referencia que un modelo “Aceptable” indica que un modelo es funcional pero básico mientras que un modelo "Bueno" corresponde a modelos competitivos en aplicaciones reales. Mientras tanto "SOTA" refleja los mejores resultados publicados en la literatura, en este caso basado en la publicación *Group DETR v2* que alcanzó un mAP50-95 de 64.5 por lo que lo tomaremos como *State of the Art*. (Chen & Wang, 2022)

A grandes rasgos observamos:

* A medida que se han aumentado la cantidad de epochs, el modelo ha obtenido mejor rendimiento.
* El **modelo 1** **de 5 epochs con solamente el dataset público** ya obtiene métricas mejores que los umbrales de referencia aceptable. Este modelo base valida que el dataset público tiene potencial y es de calidad.
* El **modelo 2 de 10 epochs con dataset público combinado con sintético** logra métricas ya superiores a umbrales de referencia bueno. Este segundo modelo valida que la inclusión del dataset sintético para aumentar la cobertura de clases poco representadas no ha distorsionado ni empeorado el rendimiento del modelo.
* El **modelo 3 de 100 epochs** obtiene valores altos equiparables o incluso superiores a los de referencia de *State of Art* mostrando un rendimiento sólido y generalizable. No obstante, se debe señalar que la comparación no es directa, dado que COCO incluye 80 clases generales, mientras que nuestro dataset contiene únicamente 30 clases de ingredientes específicos.

En base a los resultados observados, utilizaremos el modelo 3 de 100 epochs como el modelo final ya que alcanza valores muy altos en todas las métricas y demuestra capacidad para resolver de manera eficaz el problema planteado.

**Rendimiento específico de cada entrenamiento**

A graph of a function

AI-generated content may be incorrect.A continuación, ofreceremos evaluación de rendimiento más detallada de cada entrenamiento:

**A graph of a number of graphs

AI-generated content may be incorrect.Entrenamiento 1**

Podemos ver en la gráfica que:

* Las métricas de pérdida de entrenamiento y validación (box, cls, dfl) disminuyen a medida que aumenta la cantidad de epochs, esto indica que el modelo sigue aprendiendo y mejorando con más épocas de entrenamiento. Por lo tanto, en posteriores entrenamientos (modelo 2 y 3) se ha decidido aumentar la cantidad de epochs
* El rendimiento es aceptable como línea base. Precisión ~0.76 (fiable cuando predice), pero Recall bajo (~0.60), dejando escapar muchos objetos. mAP50 ~0.67 y mAP50-95 ~0.45 reflejan detecciones correctas, pero el ajuste de cajas delimitadas puede ser mejorable.
* A pesar de que el modelo tiene un rendimiento aceptable, hay clases que apenas tienen fotos para el entrenamiento como brócoli o Salchichas y hay clases que no tienen fotos en absoluto (por ejemplo salmón, sandía, lubina, etc.). Por lo tanto, para que el modelo puede predecir dichas clases, se tiene que incluir sí o sí un dataset sintético con datos sobre estas clases.

**Entrenamiento 2**

**A graph of a graph

AI-generated content may be incorrect.**De la gráfica, observamos:

* Similarmente al primer modelo, las métricas de pérdida siguen disminuyendo de manera continua a lo largo de las 10 epochs. Por lo tanto, posteriormente aumentaremos la cantidad de epochs.
* Se observa notable mejora gracias al dataset sintético en métricas de rendimiento con Precisión ~0.91 y Recall ~0.81, reduciendo tanto errores u omisiones de objetos reales. mAP50 ~0.89 y mAP50-95 ~0.65 muestran un desempeño sólido y equilibrado.
* El modelo 2 parece confundir a veces el tomate con la manzana y también el brócoli con la lechuga, probablemente debido al parecido en el color.

**A graph of a graph

AI-generated content may be incorrect.Entrenamiento 3**

* El modelo entrenado con 100 épocas muestra un aprendizaje estable y continuo sin síntomas de sobreajuste. Se decide parar el 100 epochs porque en las gráficas ya observamos como las métricas empiezan a estancarse y no hay mucha mejora incluso si aumentásemos la cantidad de epochs. Dado que los recursos computacionales son limitados, se decide dejar el modelo en 100 epochs.
* Precisión ~0.96 y Recall ~0.91, con mAP50 ~0.96 y mAP50-95 ~0.82, lo que demuestra detecciones muy precisas y cajas bien ajustadas. La matriz de confusión muestra muy pocas equivocaciones, validando al modelo como **apto para aplicaciones reales** de detección de ingredientes en imágenes de nevera.

## Ejemplo de uso del modelo final

En esta sección, se muestra un ejemplo del uso del modelo final. Se han tomado dos fotografías de una nevera real para observar el uso y predicción real del modelo. Las dos fotos han sido tomadas en ángulos diferentes y podemos observar que:

* El modelo generalmente no solo detecta todos los ingredientes en la nevera, sino que correctamente también.
* No obstante, dependiendo del ángulo observamos que no logra detectar o la lechuga o una de las mantequillas en la balda superior
* Otra cosa para destacar es que, en la primera imagen, el modelo confunde la manzana con un tomate mientras que con un mejor ángulo en la segunda imagen, el modelo logra identificarlo correctamente como manzana.

A refrigerator with food in it

AI-generated content may be incorrect.A refrigerator with different foods

AI-generated content may be incorrect.

# **Construcción del modelo NLP**

En esta sección se describe la construcción y ajuste de un modelo de procesamiento de lenguaje natural NLP capaz de detectar síntomas y las posibles deficiencias nutricionales asociadas al síntoma a partir de una descripción breve del estado de ánimo/físico del usuario. Este modelo se integraría posteriormente en el sistema de recomendación de recetas que tome en cuenta tanto los ingredientes detectados con el modelo YOLO (construido en "Creación del modelo YOLO") así como las necesidades específicas del consumidor en cuanto a deficiencia nutricional.

Un ejemplo del flujo sería para el uso del modelo NLP sería el siguiente:

1. El usuario describe "últimamente me siento muy cansado todos los días"
2. El modelo NLP detecta que el síntoma es fatiga, un síntoma usualmente asociado a posibles deficiencias de hierro, vitamina b12, folato, vitamina d y magnesio
3. Algunos de los ingredientes que podrían ayudar con las deficiencias son: carne vacuna, plátano, salchichas, huevos, etc.
4. Esto posteriormente pasaría como contexto al algoritmo de recomendación y se tiene en cuenta a la hora de priorizar que recetas recomendar

El modelo NLP permite enriquecer el sistema con información contextual sobre el estado nutricional del usuario, complementando la detección de ingredientes realizada mediante visión por computador.

## Obtención de datasets

El primer paso para la creación del modelo NLP fue la obtención de un dataset adecuado. Para entrenar un modelo capaz de predecir síntomas a partir de descripciones breves del estado físico o anímico de los usuarios, se requiere un conjunto de datos con la siguiente estructura: descripción en texto - síntoma.

No se han encontrado datos públicos disponibles con la estructura mencionada dado que la mayoría de los datos médicos están relacionados a diagnósticos clínicos y no realmente al estudio de relación entre estados y nutrición. Adicionalmente, dichos datos clínicos también son típicamente confidenciales. Debido a ello, se ha decidido crear un set de datos manualmente recopilando fuentes médicas y combinándolos con sets de datos sintéticos:

* **Fuentes médicas:** se han utilizado fuentes médicas como (Yale Medicine, s.f.) y (National Institutes of Health, s.f.) entre otrospara el entendimiento y la información de deficiencias nutricionales, así como los alimentos que podrían ayudar con los mismos.
* **Large Language Models:** se han utilizado modelos de inteligencia artificial generativa para Adicionalmente, se ha utilizado Large Language Models como Chat GPT ara generar descripciones variadas y naturales de síntomas en lenguaje cotidiano, enriqueciendo el dataset con expresiones más cercanas a como hablaría un usuario real.
* **Curación manual:** se revisaron, adaptaron y ampliaron los ejemplos para asegurar consistencia en la relación entre síntoma, deficiencia e ingredientes

El set de datos final abarca aproximadamente 38 síntomas únicas y 20 descripciones asociadas con cada síntoma. Cabe destacar que, aunque este dataset permite entrenar un modelo inicial, presenta limitaciones: las descripciones no provienen de usuarios reales, por lo que su diversidad es limitada. Futuras versiones del modelo deberían validarse o ampliarse con datos recogidos en entornos reales para mejorar el rendimiento y la representatividad del modelo.

A continuación, se muestra un ejemplo de cómo sería la estructura del set de datos:

|  |  |  |  |
| --- | --- | --- | --- |
| **Descripción** | **Síntoma** | **Posibles deficiencias** | **Ingredientes que suplen las deficiencias** |
| Últimamente me noto cansado todo el día | Fatiga | Hierro, vitamina b12, folato, vitamina d, magnesio | Carne vacuno, carne cerdo, carne pollo, carne pavo, salchichas, salmón, merluza, etc. |

## Entrenamiento del modelo

**Partición del set de datos**

Para comenzar con el entrenamiento del modelo NLP, el primer paso consiste en dividir el set de datos en subconjuntos de *train, validation y test*. Estos tres subconjuntos cumplen funciones distintas: el entrenamiento del modelo, la monitorización de las funciones de pérdida y métricas durante el aprendizaje, y la evaluación final de la calidad de modelo y su capacidad de generalización. Las ratios de la partición fueron de 72.25% para entrenamiento, 12.75% para validación y 15% para test.

Se evaluaron dos estrategias diferentes a la hora de realizar la partición del set de datos:

* **Estrategia estratificada:** la división estratificada garantiza que cada síntoma aparezca en todos los subconjuntos con cantidades consistentes y dada la estructura de nuestro set de datos (20 descripciones para 38 síntomas diferentes), también garantiza que cada síntoma aparezca al menos una vez en cada subconjunto. Esto no está garantizado por la estrategia aleatoria.
* **Forma completamente aleatoria:** se hará la división del set de datos de forma completamente aleatoria sin controlar si cada clase ha tenido representación. Puede provocar que algún síntoma no aparezca en algún subconjunto como por ejemplo validación o test.

Se entrenarán dos modelos utilizando cada tipo de estrategia y se escogerá el método que tenga mejor rendimiento.

**Estructura y flujo del entrenamiento**

El entrenamiento, independientemente de la estrategia de partición empleada, parte del modelo base *dccuchile/bert-base-spanish-wwm-uncased*, disponible en Hugging Face. Este modelo es una variante en español del modelo BERT (*Bidirectional Encoder Representations from Transformers*), un modelo de lenguaje publicado por Google en el 2018 basado únicamente en el encoder de la arquitectura Transformer. Su principal aportación es el aprendizaje de representaciones contextuales bidireccionales, de modo que cada palabra se interpreta considerando simultáneamente el contexto de la izquierda y de la derecha, lo que mejora la comprensión del texto respecto a modelos unidireccionales. (Devlin, Chang, Lee, & Toutanova, 2018)

Durante su preentrenamiento, BERT utiliza dos tareas principales: Masked Language Modeling (MLM) y, en la versión original, Next Sentence Prediction (NSP). Sobre esta base, en nuestro caso se añade una cabeza de clasificación multiclase y se ajusta el modelo mediante fine-tuning para predecir el síntoma correspondiente a partir de la descripción proporcionada por el usuario.

Dado que se prevee que el usuario final sea de habla hispana, la variante de BERT escogida, *dccuchile/bert-base-spanish-wwm-uncased,* ha sido preentrenada con un corpus en español bastante extenso, por lo que lo hace adecuada como base para aplicar transfer learning y adaptarla a nuestro problema específico: identificar síntomas a partir de descripciones breves de estados anímicos y físicos en español.

El flujo general del entrenamiento es el siguiente (para los detalles de implementación, véase anexo XXXX):

1. **Preparación de etiquetas**: Se identifican todos los síntomas únicos presentes en el dataset y se asigna a cada uno un identificador numérico. De esta forma, los etiquetas textuales (e.j: fatiga, dolor de cabeza) se transforman en números codificados (ids) que el modelo puede manejar (e.j: fatiga -> 1)
2. **Datasets y tokenización**: En primer lugar, se formatea los subconjuntos de datos a usar en *DatasetDict* ya que es la forma requerida por modelos Hugging Face. Adicionalmente cada texto se procesará con el tokenizador del modelo. Este paso convierte las frases en secuencias numéricas con un límite de longitud (256) y añade padding (relleno) o truncado si fuera necesario.
3. **Definición del modelo de clasificación:** Se instancia BERT como base, añadiendo una capa final de predicción softmax con un número de neuronas de salida que coincide con el número total de síntomas.
4. **Especificación de métricas:** Para evaluar el rendimiento se calcularán métricas estándares de clasificación multiclase como pueden ser: accuracy, F1-macro, precision-macro y recall-macro.
5. **Configuración de entrenamiento:** En este paso se establecen los parámetros de entrenamiento. A continuación, se presentan los parámetros principales, pero para la lista detallada, véase anexo XXXX.
   * La tasa de aprendizaje es 2e-5, un número estándar en el *finetuning* de los modelos
   * Las épocas establecidas para el entrenamiento son 10 con parada temprana con una paciencia de cinco. Es decir, si el modelo no mejora en 5 épocas, se detiene automáticamente
   * Se ha escogido un weight decay de 0.01 (regularización) para mitigar el posible sobre ajuste
   * Se ha establecido que al final del entrenamiento, se selecciona automáticamente el mejor checkpoint con la menor pérdida de validación.
6. **Entrenamiento y evaluación:** El modelo se entrena con los datos de train y se monitoriza con los de validation. Una vez completado el entrenamiento, se evalúa con el conjunto de test, obteniendo las métricas finales que reflejan la capacidad de generalización del modelo.

## Evaluación del modelo y ejemplos de predicción

A continuación, se detallan los resultados obtenidos durante el entrenamiento de cada modelo:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Modelo 1 (estratificado)** | | | **Modelo 2 (aleatorio)** | | |
| **Epoch** | **Training Loss** | **Val Loss** | **Accuracy** | **Training Loss** | **Val Loss** | **Accuracy** |
| 1 | 3.54 | 3.49 | 0.13 | 3.54 | 3.40 | 0.16 |
| 2 | 3.18 | 3.03 | 0.63 | 2.93 | 2.89 | 0.59 |
| 3 | 2.60 | 2.51 | 0.81 | 2.21 | 2.33 | 0.81 |
| 4 | 2.24 | 1.98 | 0.90 | 2.29 | 1.85 | 0.89 |
| 5 | 1.81 | 1.54 | 0.94 | 1.22 | 1.42 | 0.93 |
| 6 | 1.11 | 1.21 | 0.96 | 1.32 | 1.14 | 0.93 |
| 7 | 1.00 | 0.97 | 0.97 | 0.96 | 0.92 | 0.97 |
| 8 | 1.12 | 0.83 | 0.97 | 0.85 | 0.78 | 0.99 |
| 9 | 0.82 | 0.73 | 0.97 | 0.65 | 0.70 | 0.99 |
| 10 | 0.76 | 0.71 | 0.97 | 0.66 | 0.67 | 0.99 |

Durante el entrenamiento, se compararon dos modelos con distinta estrategia de partición del dataset: estratificada y aleatoria. Ambos alcanzaron resultados muy altos con diferencias mínimas.

**Similitudes principales**

* El rendimiento en validación se estabiliza rápidamente en pocas épocas.
* Se observan métricas finales muy buenas con altos valores en accuracy, F1, precisión y recall
* Los resultados en test sugieren un nivel de generalización elevado
* Se detecta un posible riesgo de data leakage debido a la similitud entre ejemplos parafraseados en los diferentes conjuntos.

**Diferencias observadas**

* El modelo de estrategia de partición aleatorio obtiene ligeramente mejor rendimiento en el entrenamiento, aunque las diferencias son muy ligeras.
* El estratificado converge antes (a partir de época 6) con valores de accuracy estabilizándose, mientras que el aleatorio se estabiliza más tarde (a partir de época 8).

A screenshot of a graph

AI-generated content may be incorrect.A continuación, también detallamos los valores obtenidos en las métricas clave al evaluar el modelo con el set de prueba:

|  |  |  |
| --- | --- | --- |
|  | **Modelo 1 (estratificado)** | **Modelo 2 (aleatorio)** |
| Pérdida de evaluación | 0.699 | 0.63 |
| Accuracy | 1 | 0.982 |
| F1 Macro | 1 | 0.98 |
| Precision | 1 | 0.986 |
| Recall Macro | 1 | 0.98 |

Se observa que ambos modelos han obtenido muy buenos. El modelo estratificado alcanzó métricas perfectas (accuracy y F1 = 1.0) y una matriz de confusión con diagonal perfecta. El modelo aleatorio mostró ligeras confusiones (ej. confundir calambres menstruales con a calambres nocturnos), pero con una pérdida menor y resultados igualmente muy altos.

A pesar de los buenos resultados, estos valores tan elevados son poco habituales y probablemente reflejan cierto grado de *data leakage* (fuga de datos). El dataset incluye descripciones generadas y parafraseadas que pueden ser muy similares entre sí. Esto incrementa el riesgo de “fuga de información” entre train/validation/test dado que el modelo está entrenándose con descripciones similares a los que hay en la validación o test. Esto puede inflar artificialmente las métricas, un efecto agravado por el tamaño reducido del conjunto de datos, y comprometer la robustez y la capacidad de generalización del modelo

No obstante, debemos considerar la relevancia práctica también. En el caso de uso planteado, es esperable que las descripciones reales de los usuarios se asemejen a los ejemplos de entrenamiento, ya que suelen seguir patrones lingüísticos recurrentes (“me siento cansado todo el día”, “dolor menstrual intenso”, etc.). En este sentido, un alto rendimiento en frases similares no es necesariamente negativo: refleja de manera realista cómo se comportaría el modelo en producción. Aunque cada usuario puede expresarse de formas distintas, es muy probable que utilicen sinónimos o construcciones cercanas a las vistas en el entrenamiento, por lo que cierto grado de solapamiento resulta inevitable.

## En conclusión, para un primer prototipo se selecciona el modelo con partición aleatoria como modelo final, ya que presenta una pérdida inferior, aunque con métricas ligeramente más bajas en accuracy y recall. Se destaca, no obstante, la necesidad de ampliar y actualizar el dataset con descripciones reales de usuarios a medida que la aplicación se utilice y puede coleccionas más datos, con el fin de reforzar la robustez y validez del modelo en escenarios más diversos.

## Ejemplo de uso del modelo final

En esta sección se ilustra con un ejemplo práctico el funcionamiento del modelo NLP y se verifica su capacidad de predecir correctamente nuevas descripciones en un escenario de uso real. Se observan los siguientes resultados:

|  |  |  |  |
| --- | --- | --- | --- |
| **Texto** | **Síntoma real** | **Predicción del modelo** | **Confianza** |
| Últimamente me siento agotado todo el día, aunque duerma bien. | Fatiga | fatiga | 0.32 |
| Tengo cólicos muy fuertes cada mes antes de mi periodo. | Calambres menstruales | calambres menstruales | 0.26 |
| Mi piel está áspera y se descama con facilidad. | Piel seca | piel seca | 0.60 |

Se puede observar que el modelo predice correctamente todos los síntomas, aunque en algunos casos la confianza del modelo es relativamente baja. Una vez el modelo predice el síntoma, el sistema sigue este flujo:

1. El modelo determina el síntoma predicho (ej. fatiga) y lo vincula con una lista de posibles deficiencias nutricionales relacionadas (hierro, vitamina B12, folato, vitamina D y magnesio).
2. A partir de estas deficiencias, y considerando las 30 clases de ingredientes definidas previamente, se seleccionan aquellos alimentos que pueden contribuir a suplirlas (por ejemplo, plátano para magnesio o carnes para vitamina B12).

De esta manera, el modelo NLP no solo reconoce el síntoma descrito por el usuario, sino que también proporciona el contexto nutricional necesario para integrarse posteriormente en el sistema de recomendación de recetas.

**Algoritmo de recomendación y RAG LLM como juez**

Tal y como se ha explicado en la sección *2. Flujo de la aplicación y algoritmos utilizados,* para poder recomendar recetas personalizadas en base a los inputs introducidos por el usuario (capturados e interpretados por los modelos YOLO y NLP construidos previamente), se necesita un sistema de puntuación para poder decidir qué recetas serían las ideales para el usuario. En esta sección, se detallarán los dos últimos pasos en la estructura del recomendador inteligente de recetas que cumplen con esta función, el sistema de puntuación en base a métricas definidas y RAG LLM que actúa como juez que toma la decisión final.

## **Preprocesamiento y estandarización del dataset de recetas**

Antes de detallar el algoritmo de recomendación, es necesario describir las fases de preprocesamiento que permiten integrar los modelos previos y preparar el dataset de recetas para su uso en la aplicación.

1. **Integración de los modelos YOLO y NLP**: Una vez construidos los modelos de YOLO y NLP, es necesario integrarlos para el uso real. Esto implica unificar las salidas: por un lado, la lista de ingredientes detectados en la nevera (YOLO), y por otro, los síntomas y posibles deficiencias nutricionales identificados a partir de la descripción del usuario (NLP). Estos resultados se combinan y se convertirán como contexto inicial para el algoritmo de recomendación.
2. **Obtención y traducción del dataset de recetas**: Para la recomendación se utilizó el dataset público *RecipeNLG* que contiene más de 2 millones de recetas en inglés. Sin embargo, dado que el público objetivo es hispanohablante, se procedió a la estandarización y traducción de las recetas al español.

El proceso completo de normalización y traducción de forma resumida incluyó:

1. **Construcción de un diccionario de equivalencias:** Se creó un diccionario manual de mapeo entre ingredientes en inglés y su equivalente en español principalmente para agrupar diferentes variantes en inglés y mapear a un único término en español. Por ejemplo: ["potato", "russet", "yukon gold"] → patata`
2. **Normalización de términos:** Se aplicaron transformaciones textuales básicas para asegurar uniformidad en el dataset. Esto incluye conversión de todos los términos a minúsculas, eliminación de caracteres especiales, traducción directa de cada término por su equivalente en español si existe en el diccionario de mapeo. Se tradujeron 43584 palabras directamente con el mapeo.
3. **Traducción automática con MarianMT:** los términos no contemplados en el diccionario se tradujeron automáticamente mediante el modelo MarianMT a gran escala. MarianMT es una familia de modelos de traducción automática neural desarollada por Microsoft basada en la arquitectura Transformer (Junczys-Dowmunt, 2018). Se tradujeron en total 152434 ingredientes únicos con el modelo MarianMT.

## **Algoritmo de recomendación por reglas y puntuación**

Para que la aplicación inteligente pueda seleccionar recetas desde el dataset de más de 2 millones de recetas, se definieron métricas específicas que permiten construir un algoritmo de recomendación basado en reglas de puntuación. Estas métricas puntúan cada receta en función del contexto recibido (ingredientes detectados en la nevera por el modelo YOLO y síntomas/deficiencias nutricionales identificados por el modelo NLP).

***Notación:***

* *: representa el conjunto de ingredientes de la receta*
* *: representa el conjunto de ingredientes de la nevera detectados por YOLO*
* *: representa el conjunto de ingredientes que ayudan a cubrir las deficiencias relacionadas con el síntoma predicho por el modelo NLP*
* *: representa el número de tokens/palabras del texto de instrucciones de la receta*

***Métricas definidas:***

**1) Similitud de Jaccard (Receta vs Nevera):**

Mide el solapamiento entre los ingredientes necesarios para la receta y los disponibles en la nevera. Su rango es [0,1], donde valores altos indican mayor compatibilidad. Se asigna un peso positivo de 0.5 para el cálculo final de la puntuación dada la relevancia en el contexto de minimizar el desperdicio alimenticio.

**2) Penalización por ingredientes faltantes**

Cuenta cuántos ingredientes de la receta no están en la nevera, normalizado por el tamaño de la receta. Su rango es de [0,1], los valores bajos son mejores. Se asigna un peso negativo de 0.25 para el cálculo final de la puntuación.

**3) Cobertura de deficiencias**

Es una métrica que calcula la cobertura de las deficiencias nutricionales. Mide de los ingredientes que ayudan a cubrir las deficiencias (H), cuántos aparecen en la receta (R). Se normaliza por (H). Su rango es de [0,1]. Cuanto más alto sea el valor, mejor es. Para la fórmula final se decide asignar un peso positivo del 0.2.

**4) Esfuerzo necesitado**

Dado que no hay realmente ningún indicador directo de esfuerzo, se asume que las recetas con instrucciones más largas implican mayor esfuerzo. Por lo tanto, se normaliza la cantidad de tokens de texto de las recetas con min-max de forma que la receta más corta sea 0 y la más larga 1. Así podemos medir o cuantificar el esfuerzo de una receta. El rango es del 0 al 1. Cuanto más bajo sea el valor, mejor es. Para la fórmula final se decide asignar un peso negativo del 0.1.

La fórmula final que utilizará el sistema para puntuar cada receta en el dataset de recetas será la siguiente:

El sistema selecciona las 10 recetas con mayor puntuación acumulada como candidatas para la fase final de recomendación.

## **Uso de LLM con RAG como juez final y prompt engineering**

En la etapa final de la recomendación, se incorpora un modelo de lenguaje de gran tamaño (LLM) bajo el esquema de Retrieval-Augmented Generation (RAG). Este LLM actúa como juez, seleccionando de entre las diez recetas mejor puntuadas las tres más adecuadas para el usuario, y proporcionando además una explicación contextual.

El RAG es una técnica que combina dos componentes:

* **Recuperación de información (retrieval):** el LLM antes de responder al usuario consultará bases de datos o información interna para obtener contexto. En este caso, los datos internos son las diez recetas con mejor puntuación del algoritmo basado en reglas y el contexto específico son las predicciones de YOLO (ingredientes disponibles en la nevera) y NLP (el síntoma predecido con sus posibles deficiencias nutricionales)
* **Generación de texto:** el LLM utiliza la información recuperada para producir una respuesta final.

Para el prototipo se emplea el LLM “*llama3.1:8b”* ejecutado mediante Ollama, escogido principalmente porque no tiene coste, dado que se ejecuta en local, y por su su equilibrio entre capacidad de razonamiento y eficiencia computacional. Los modelos llaman forman parte de una familia de modelos de lenguaje desarrollados for *Meta*. Están basados en la arquitectura transformer y entrenados con grandes cantidades de datos multilingües. (Meta, s.f.) El modelo utilizado contiene aproximadamente 8.03 mil millones de parámetros y hasta 128 mil tokens.

Para poder controlar el formato, la salida, así como el comportamiento del LLM, se puede llevar a cabo técnicas de prompt engineering, en particular a través de dos componentes principales:

* **System Prompt (instrucciones globales):** este prompt esta establece restricciones y reglas generales que debe seguir el modelo. En nuestro caso, se ha especificado que el modelo debe:
  + utilizar exclusivamente la información del contexto proporcionado, evitando generar información no disponible que no tiene base (alucinaciones)
  + descartar recetas que no son apropiadas para el consumo (ej. Mascarillas de pelo con ingredientes alimenticios u otras recetas que no son para el consumo)
  + verificar siempre la viabilidad de la receta en función de los ingredientes detectados en la nevera
  + comprobar que la receta contribuye a suplir la deficiencia nutricional asociada al síntoma detectado

A black background with text

AI-generated content may be incorrect.

* **User Prompt (instrucciones específicas):** este prompt define el formato de la salida esperada. El modelo debe:
  + dirigirse directamente al usuario (tutear) e informar del síntoma detectado y sus posibles deficiencias, recordando que siempre es mejor consultar un médico dado que la recomendación de recetas son orientaciones
  + enumerar los ingredientes disponibles en la nevera detectados
  + finalmente recomendar tres recetas, indicando para cada una: el nombre, los ingredientes requeridos (traducidos al español), una breve justificación de cómo ayuda al síntoma, y posibles sustituciones cuando falten ingredientes
  + describir las instrucciones de preparación en base al texto de la receta original.

A black background with many small lights

AI-generated content may be incorrect.

De este modo, el algoritmo de puntuación hace un primer filtro reduciendo 2 millones de recetas a 10 con mejor puntuación y el LLM con RAG revisa esas 10 recetas siguiendo unas instrucciones y aplica “razonamiento” para elegir las 3 recetas mejores finales. Cabe destacar que el modelo LLM se usa con temperatura 0.1 (rango 0-1) para que siempre de una respuesta consistente sin mucha variación.

**Productivizar el sistema**

**Conclusiones y futuras mejoras**

**Resumen**

**Referencias**

ANEXO

A screen shot of a computer screen

AI-generated content may be incorrect.A group of graphs on a white background

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect. A group of colorful bars

AI-generated content may be incorrect.