



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

**Proyecto Fin de Máster**

Máster en Big Data, Inteligencia Artificial y Data Science

**HealthBite:**

Aplicación con Visión Artificial, Lenguaje Natural y LLMs para la personalización nutricional a partir de ingredientes y estado del usuario

Septiembre 2025

**Autor:** Óscar Xu Zhou

# Tabla de contenidos

Resumen .....	1
1. Introducción .....	2
1.1 Contexto y justificación del proyecto.....	2
1.2 Objetivos del proyecto.....	2
2. Flujo de la aplicación y algoritmos utilizados .....	3
3. Análisis exploratorio de datos (EDA) .....	4
3.1 Obtención de dataset: .....	4
3.2 Depuración y procesamiento realizado:.....	4
3.3 Análisis de datos: .....	5
.....	6
4. Creación del modelo YOLO .....	6
4.1 Obtención de datasets .....	6
4.2 Entrenamiento del modelo .....	8
4.3 Evaluación del modelo .....	8
4.4 Ejemplo de uso del modelo final .....	11
5. Construcción del modelo NLP .....	11
5.1 Obtención de datasets .....	12
5.2 Entrenamiento del modelo .....	12
5.3 Evaluación del modelo .....	14
5.4 Ejemplo de uso del modelo final NLP .....	15
6. Algoritmo de recomendación y LLM juez con RAG .....	16
6.1 Preprocesamiento y estandarización del dataset de recetas .....	16
6.2 Algoritmo de recomendación por reglas y puntuación .....	16
6.3 Uso de LLM con RAG como juez final y prompt engineering .....	17
7. Productivización del sistema .....	18
8. Conclusiones y resultados .....	19
8.1 Limitaciones encontradas y futuras mejoras.....	20
Anexo .....	21
Anexo A: Código y gráficas del análisis exploratorio de datos .....	21
Anexo B: Código y fotografías ejemplo del set de datos del modelo YOLO .....	22
Anexo C: Entrenamiento del modelo NLP .....	24
Anexo D: Traducción del dataset de recetas .....	25
Anexo E: Construcción del algoritmo de puntuación y LLM con RAG como juez .....	26
Anexo F: Código completo de las funciones e interfaz de la aplicación .....	26
Bibliografía .....	28

# Resumen

Este proyecto se centra en el desarrollo de HealthBite, una aplicación inteligente que combina visión artificial, procesamiento de lenguaje natural y modelos de lenguaje de gran escala (LLMs) para ofrecer recomendaciones de recetas personalizadas. La propuesta introduce tres factores innovadores principalmente: (1) la capacidad de detectar automáticamente los alimentos disponibles en la nevera del usuario, (2) la opción de inferir posibles síntomas o deficiencias nutricionales a partir de una breve descripción de estado anímico o físico y (3) una comunicación transparente e interactiva con el usuario utilizando la tecnología de LLMs que permite al usuario comprender la razón detrás de cada recomendación.

El objetivo principal del proyecto es reducir el desperdicio alimenticio, priorizando la recomendación de recetas que utilicen ingredientes ya disponibles y rescatando productos olvidados o escondidos en la nevera que podrían caducar y deshacerse, contribuyendo así a disminuir la huella de carbono individual. Al mismo tiempo, la aplicación busca mejorar la salud preventiva mediante las sugerencias de recetas adaptadas que puedan suplir posibles deficiencias en nutrientes inferidas a partir de las descripciones del usuario.

El flujo de uso de la aplicación es sencillo para el usuario: basta con describir su estado anímico o físico y subir una fotografía de su nevera. Con esta información, intervienen de forma integrada los distintos modelos. En primer lugar, un modelo de visión artificial (YOLO), entrenado con más de 17 000 imágenes correspondientes a 30 clases únicas de alimentos, identifica los ingredientes disponibles. Paralelamente, un modelo de procesamiento de lenguaje natural (NLP), entrenado con descripciones en español de estados anímicos y físicos, infiere y predice posibles síntomas junto con las deficiencias nutricionales asociadas. Los resultados de ambos modelos se combinan en un algoritmo de puntuación que evalúa un conjunto de más de dos millones de recetas, priorizando aquellas que mejor se ajustan al contexto específico del usuario. Finalmente, una capa adicional basada en un LLM con RAG revisa las recetas mejor puntuadas y actúa como agente de razonamiento: no solo selecciona las opciones definitivas que se recomendarán, sino que también proporciona explicaciones claras y justificadas sobre por qué fueron elegidas y cómo responden a las necesidades particulares del individuo. Durante el desarrollo, se comprobó que los modelos construidos alcanzan un rendimiento sólido y generan predicciones fiables, validando así la viabilidad del sistema propuesto.

De esta forma, HealthBite ofrece un valor diferencial frente a las aplicaciones existentes en el mercado, que mayoritariamente suelen limitarse al escaneo de productos mediante código de barras o a la recomendación de dietas genéricas (generalmente de carácter saludable y poco adaptadas), sin tener en cuenta el contexto real del usuario.

El trabajo también identifica diversas áreas de mejora futura con potencial: la ampliación del número de ingredientes reconocidos, la incorporación progresiva de descripciones de estado reales para robustecer el modelo de lenguaje, el despliegue en la nube para facilitar el acceso, etc. Especialmente, se destaca la posibilidad de integración de un sistema de registro de datos de las descripciones de los usuarios y las posibles deficiencias identificadas en una base de datos SQL. Esta base alimentará y actualizará automáticamente un dashboard visual que permitiría analizar tendencias y descubrir patrones de salud a largo plazo que, de otro modo, podrían pasar inadvertidos.

En conjunto, HealthBite constituye una prueba de concepto sólida que demuestra cómo la inteligencia artificial puede aplicarse de forma innovadora a la nutrición personalizada, la sostenibilidad y la salud preventiva, con un alto potencial de impacto social y práctico.

# 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 de la producción alimenticia, 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 medioambiental, 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

Se detallan los objetivos del proyecto a continuación:

### Objetivo principal:

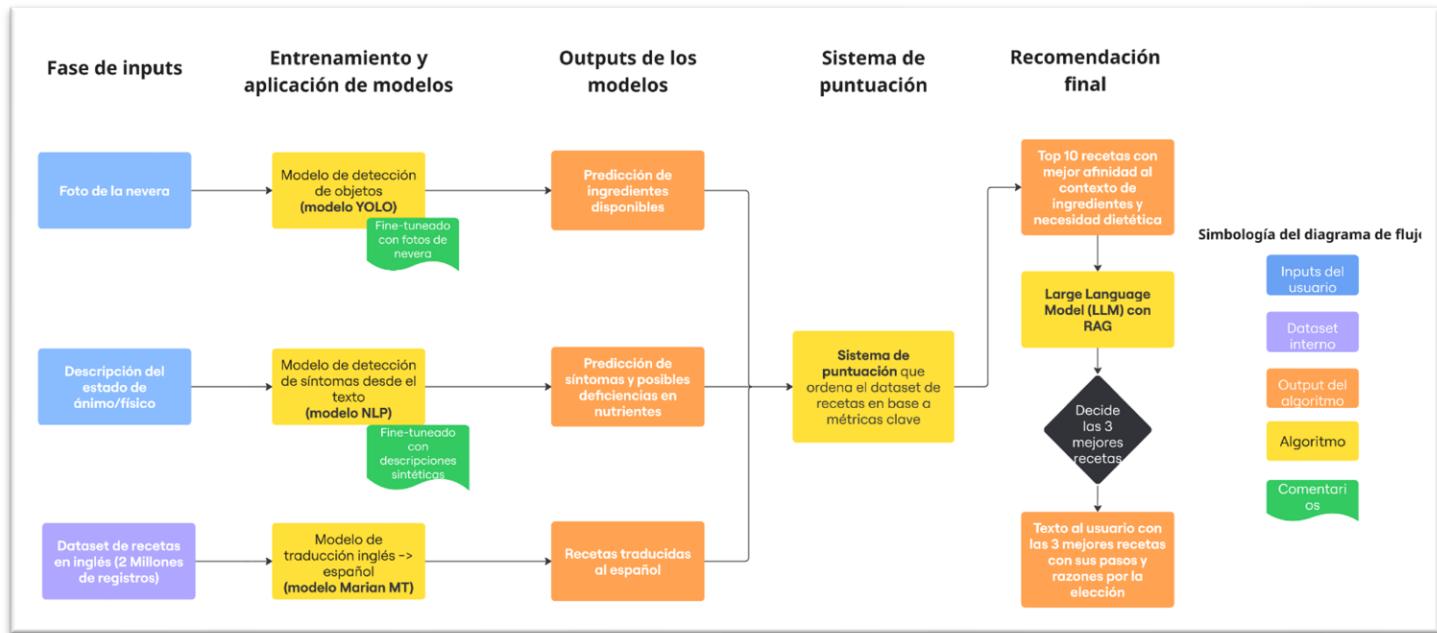
Desarrollar una aplicación inteligente capaz de recomendar recetas considerando tanto los ingredientes disponibles en la nevera del usuario como sus posibles deficiencias nutricionales, inferidas a partir de descripciones de su estado anímico o físico. Adicionalmente, será capaz de explicar las razones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

### Objetivos específicos:

- Realizar un análisis exploratorio de datos (EDA) sobre resultados de encuestas de consumo 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 de 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, entre otros factores).

- 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.

## 2. Flujo de la aplicación y algoritmos utilizados



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.
- También se carga internamente un dataset de recetas (2+ millones de registros) que servirá como base para las recomendaciones.

### 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 e inferir a partir de una descripción síntomas y posibles deficiencias nutricionales. Se ha fine-tuneado con descripciones anímico-físicas con un dataset sintético compilado a partir de fuentes médicas, clínicas y generaciones artificiales.
- 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 usando NLP)
- Recetas en español (a partir de la traducción masiva usando Marian MT)

Estos resultados alimentarán el sistema de puntuación.

## **Sistema de puntuación:**

Se ha desarrollado un algoritmo que puntuá 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 puntuada 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.

## **3. 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 tienen mayor consumo en la sociedad española. Dado que la aplicación incorporará un modelo YOLO para reconocer ingredientes en imágenes de neveras, es necesario priorizar un número manejable de clases. Como las posibles combinaciones de ingredientes en la nevera son prácticamente infinitas, entrenar el modelo sin restricciones implicaría costes computacionales muy elevados. Por ello, se ha decidido fijar un tope de 30 clases a entrenar y predecir para equilibrar cobertura y coste computacional. Por este motivo, se realiza un EDA en los 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.

### **3.1 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 *Anexo A: Código y gráficas del análisis exploratorio de datos*). El dataset incluye métricas como consumo per cápita, gasto per cápita y penetración de categoría (porcentaje de hogares consumidores).

### **3.2 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").

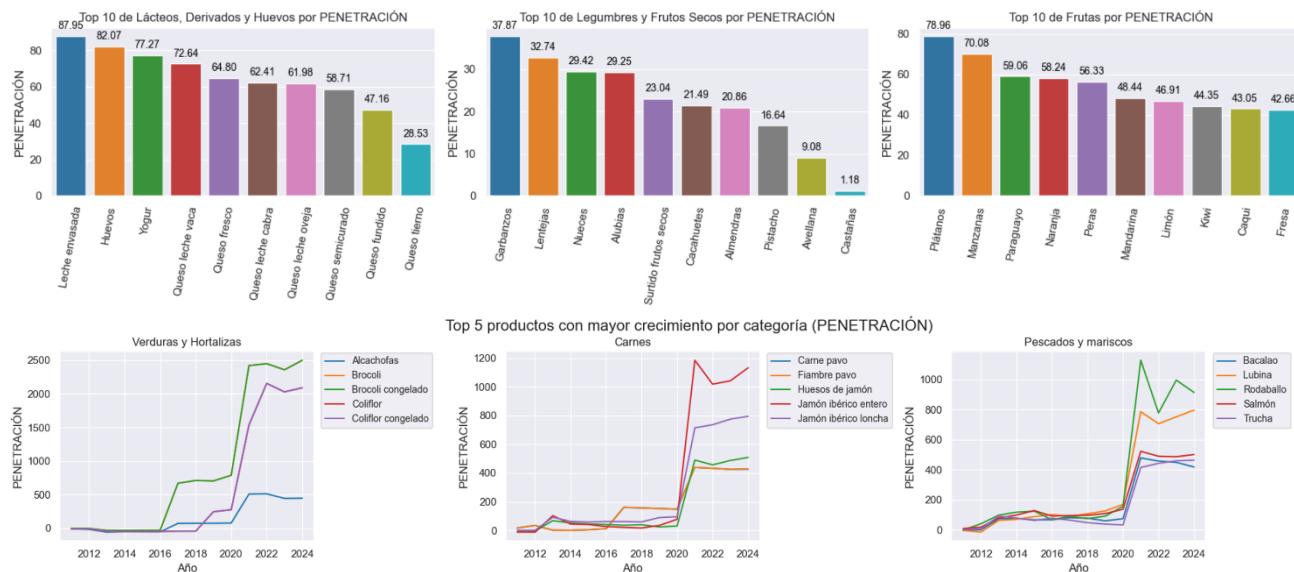
En segundo lugar, se refinan algunos nombres para mejor legibilidad (ej: T.Huevos UNDS -> "Huevos"). Por último, dado que hay demasiados nombres únicos, realizar un análisis exploratorio en ellos sería impráctico. Para ello, se ha decidido agrupar diferentes alimentos utilizando categorías de la pirámide alimenticia.

Se muestran 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

### 3.3 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 recortadas (para las gráficas completas, véase Anexo A: Código y gráficas del análisis exploratorio de datos).



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 número en 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 almacenarán en la nevera (e.j: conservas no abiertas de lata, pasta, etc.). Estos alimentos tienen baja prioridad.
- **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:

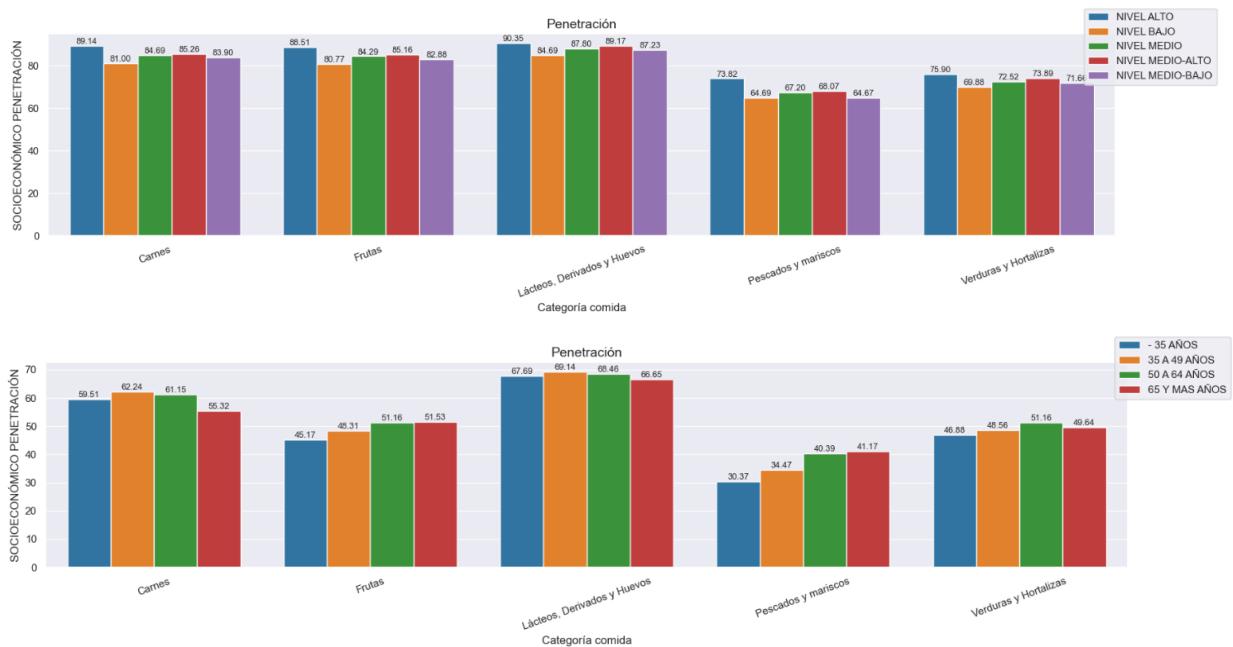
Verduras y hortalizas	Pescados y mariscos	Lácteos, derivados y huevos	Carnes	Frutas
Tomate	Merluza	Leche	Carne pollo	Plátano
Cebolla	Gambas/Langostinos	Huevos	Carne cerdo	Aguacate
Patata	Mix marisco/molusco	Yogur	Carne vacuno	Sandía
Lechuga/Endivia	Lubina	Queso	Salchichas	Limón
Zanahorias	Salmón	Mantequilla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

Pepino			
Champiñones			
Brocoli			
Coliflor			

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 dependiendo de factores demográficos. 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.



## 4. 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 artificial basado en redes neuronales convolucionales que, en una única pasada por la imagen, predice y devuelve simultáneamente cajas delimitadoras, la confianza y clase de cada objeto, permitiendo detección en tiempo real. (Redmon, 2015)

### 4.1 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 recopilado diversos datasets con fotografías de ingredientes en el interior de la nevera, obtenidos principalmente de la plataforma *Roboflow*, de características similares a Kaggle. Los diferentes datasets públicos han sido unificados en uno definitivo. Antes de ello, se llevó a cabo una estandarización del mapeo de clases, con el fin de garantizar la consistencia en las etiquetas empleadas en los distintos conjuntos de datos.

A pesar de haber datasets públicos, no se disponen de fotografías de la totalidad de 30 clases únicas 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. Con esta técnica se logra complementar los datasets públicos, incorporando aquellas clases de ingredientes que no estaban representadas en las imágenes originales.

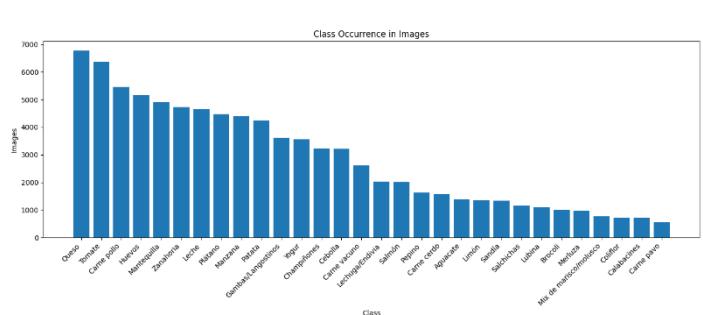
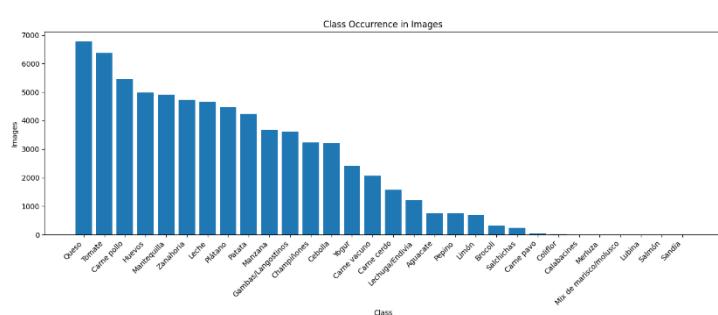
Se muestra a continuación una representación gráfica del proceso de generación de datos sintéticos.

- En primer lugar, se obtienen 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 obtienen fotos de neveras vacías para utilizar como fondos.
- 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 Anexo B: Código y fotografías ejemplo del set de datos del modelo YOLO) toma diferentes ingredientes sin fondo y los posiciona aleatoriamente en las áreas delimitadas de diferentes fotos de nevera para simular fotos realistas.



- Por último, para aumentar la diversidad del dataset sintético, se utiliza una función 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.

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.

## 4.2 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, se trabajará en reentrenar el modelo base aplicando transfer learning para poder adaptarlo a nuestro problema y contexto específicos, la detección de diferentes 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 (para más detalle sobre parámetros y argumentos, véase: Anexo B: Código y fotografías ejemplo del set de datos del modelo YOLO). 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 subconjunto de entrenamiento permite el aprendizaje al 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 del modelo sobre datos no vistos.

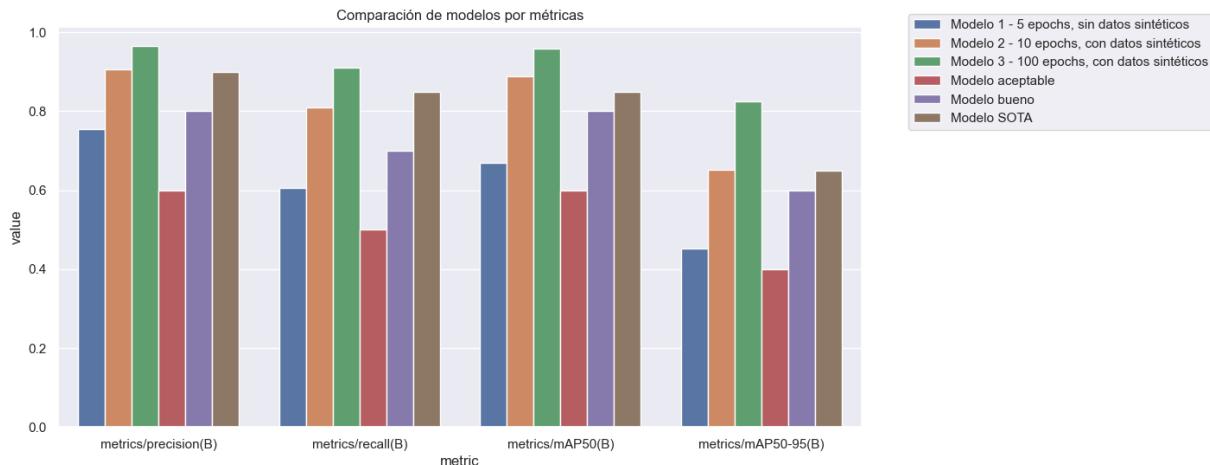
### Modelos entrenados

- **Modelo 1 - 5 epochs sin datos sintéticos:** Se ha entrenado un modelo base sencillo de tan solo 5 epochs (*donde cada epoch corresponde a una pasada completa sobre el conjunto de datos de entrenamiento*). En este modelo se utilizaron exclusivamente los datasets públicos descargados, sin incluir el conjunto sintético generado. El objetivo fue **validar la calidad inicial de los datos públicos**, aun cuando la baja representación de ciertas clases limita el rendimiento alcanzado por el modelo.
- **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:** El modelo final se entrenó durante más iteraciones y con una resolución superior a la estándar (*768x768 píxeles en lugar de 640*). Esta configuración busca mejorar la detección de objetos pequeños a costa de un mayor consumo de recursos y menor velocidad de inferencia.

## 4.3 Evaluación del modelo

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



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:  $TP/(TP + FP)$
- **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:  $TP/(TP + FN)$
- **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 más estricta.

Para fines comparativos, se establecen umbrales de referencia basados en valores reportados habitualmente en la literatura para entender si nuestro modelo alcanza buenos valores en las métricas. 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 toma como referencia que un modelo considerado “Aceptable” corresponde a un sistema funcional pero básico, mientras que un modelo “Bueno” describe 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:

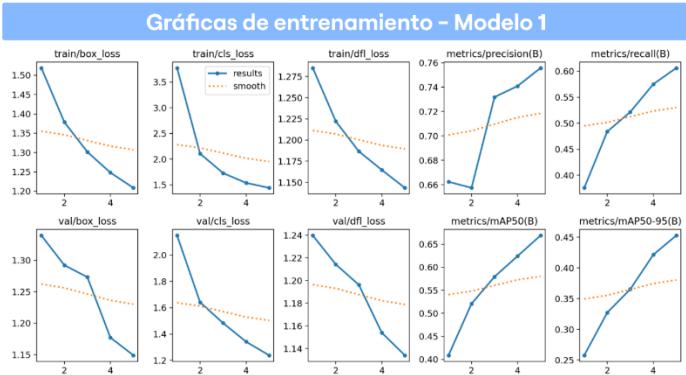
- A medida que se han aumentado la cantidad de epochs, el modelo ha obtenido mejor rendimiento.
- El **modelo 1 de 5 epochs con solamente el dataset público** ya alcanza métricas superiores a los umbrales de referencia definidos como “Aceptable”. Este modelo base confirma que el dataset público presenta un buen potencial y calidad suficiente para el entrenamiento.
- El **modelo 2 de 10 epochs de dataset público combinado con sintético** obtiene métricas que superan los 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 continuación, ofreceremos una evaluación de rendimiento más detallada de cada entrenamiento:

#### Entrenamiento 1

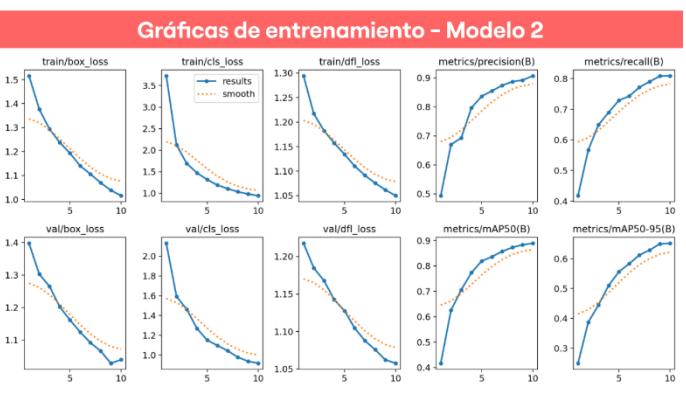


Podemos ver en la gráfica que:

- 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. Las métricas 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 representación en absoluto (por ejemplo, salmón, sandía, lubina, etc.). Por lo tanto, para que el modelo pueda predecir dichas clases, resulta imprescindible incluir un dataset sintético con datos sobre estas clases.

#### Entrenamiento 2



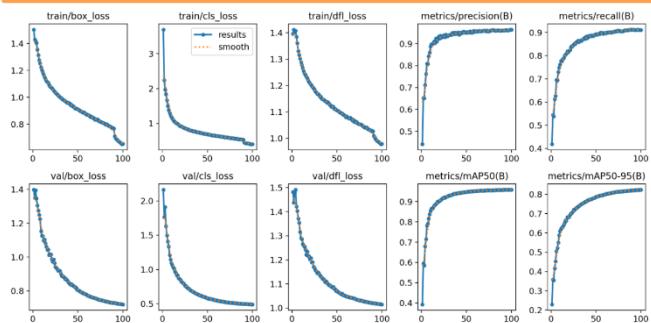
De la gráfica, observamos:

- 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 y aumento de epochs 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.

### Entrenamiento 3

Gráficas de entrenamiento - Modelo 3



- El modelo entrenado con 100 epochs muestra un aprendizaje estable y continuo sin síntomas de sobreajuste. Se decide parar aproximadamente en 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.

- Las métricas: Precisión ~0.96 y Recall ~0.91, con mAP50 ~0.96 y mAP50-95 ~0.82, demuestran que las detecciones son muy precisas y cajas bien ajustadas. La matriz de confusión muestra muy pocas equivocaciones, validando el modelo como apto para aplicaciones reales de detección de ingredientes en imágenes de nevera. (Para la gráfica de matriz de confusión, véase Anexo B: Código y fotografías ejemplo del set de datos del modelo YOLO)

#### 4.4 Ejemplo de uso del modelo final

En esta sección, se presenta un ejemplo práctico del modelo final. Se han tomado dos fotografías de una nevera real para evaluar el funcionamiento y predicción 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.
- Cabe destacar 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.



## 5. 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 "4. Creación del modelo YOLO") así como las necesidades específicas del consumidor en cuanto a deficiencia nutricional.

Un ejemplo del flujo de 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 el sistema lo tendrá en cuenta a la hora de priorizar qué 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 artificial.

## 5.1 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 → posibles deficiencias nutricionales → posibles ingredientes que cubrirían las deficiencias.

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 son también 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 otros para el entendimiento y obtención de información de deficiencias nutricionales, así como los alimentos que podrían ayudar con los mismos.
- **Large Language Models:** se utilizaron modelos de inteligencia artificial generativa, como *ChatGPT*, para producir descripciones variadas y naturales de síntomas en lenguaje cotidiano, enriqueciendo así el *dataset* con expresiones más cercanas a las que emplearía un usuario real.
- **Curación manual:** la información obtenida fue revisada, adaptada y ampliada de manera manual para asegurar su coherencia y calidad.

El set de datos final abarca aproximadamente 38 síntomas únicos y 20 descripciones asociadas a 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.

## 5.2 Entrenamiento del modelo

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:** garantiza que cada síntoma esté representado en todos los subconjuntos (entrenamiento, validación y prueba) con cantidades proporcionales. En nuestro caso, dado que el conjunto incluye 20 descripciones para 38 síntomas distintos, esta estrategia asegura que cada síntoma aparezca al menos una vez en cada subconjunto, algo que no se garantiza con una división aleatoria.
- **Forma completamente aleatoria:** se hará la división del set de datos de forma completamente aleatoria sin controlar si cada clase ha tenido representación. Esto puede ocasionar que algunos síntomas no aparezcan en determinados subconjuntos, como validación o prueba.

Se entrenarán dos modelos utilizando cada tipo de estrategia y se escogerá un modelo final.

### **Estructura y flujo del entrenamiento**

El entrenamiento, independientemente de la estrategia de partición empleada, parte del modelo base *dcuuchile/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 prevé que el usuario final sea de habla hispana, la variante de BERT escogida, *dcuuchile/bert-base-spanish-wwm-uncased*, ha sido preentrenada con un corpus en español bastante extenso, por lo que la 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:

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, las etiquetas textuales (e.j: fatiga, dolor de cabeza) se transforman en números codificados (ids) que el modelo puede manejar (e.j: fatiga → 1, dolor de cabeza → 2)
2. **Datasets y tokenización:** En primer lugar, se formatean los subconjuntos de datos a usar en *DatasetDict* ya que es la forma estándar 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 C: *Entrenamiento del modelo NLP*.
  - La tasa de aprendizaje es 2e-5, un valor común de base en el *finetuning* de los modelos NLP.

- 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.

### 5.3 Evaluación del modelo

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

Epoch	Modelo 1 (estratificado)			Modelo 2 (aleatorio)		
	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.

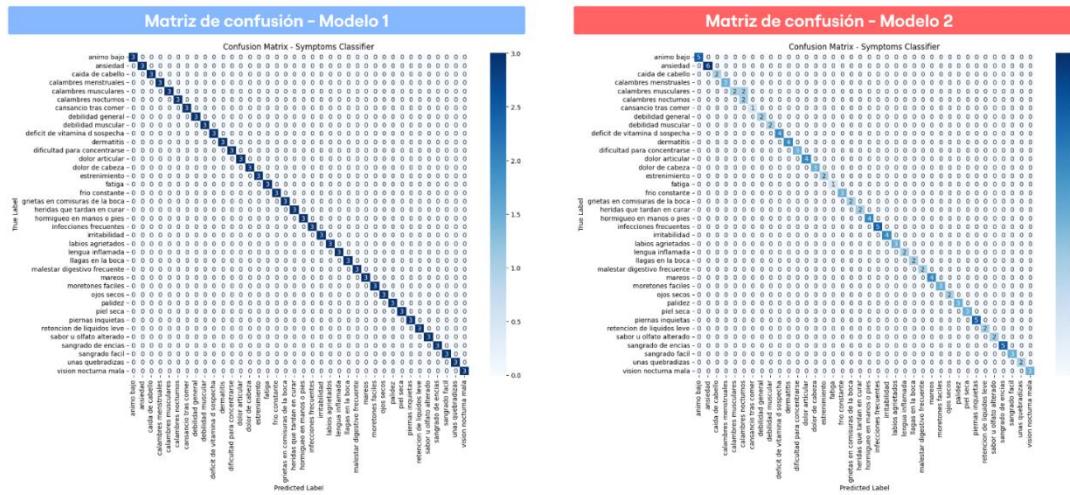
#### Diferencias observadas

- El modelo de estrategia de partición aleatoria 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 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 valores 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 demasiado similares a las 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 puede 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: puede reflejar 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 colecciónar más datos, con el fin de reforzar la robustez y validez del modelo en escenarios más diversos.

#### 5.4 Ejemplo de uso del modelo final NLP

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.

## 6. Algoritmo de recomendación y LLM juez con RAG

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, el sistema de puntuación en base a métricas definidas y LLM con RAG que actúa como juez que toma la decisión final.

### 6.1 Preprocesamiento y estandarización del dataset de recetas

Antes de detallar el algoritmo de recomendación, es necesario describir las fases de preprocesamiento que han permitido 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 (para el código, véase *Anexo D: Traducción del dataset de recetas*):

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 mediante el modelo MarianMT a gran escala. MarianMT es una familia de modelos de traducción automática neural desarrollada por Microsoft basada en la arquitectura Transformer (Junczys-Dowmunt, 2018). Se tradujeron en total 152434 ingredientes únicos con el modelo MarianMT.

### 6.2 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:

- *R* : representa el conjunto de ingredientes de la receta
- *F* : representa el conjunto de ingredientes de la nevera detectados por YOLO
- *H* : representa el conjunto de ingredientes que ayudan a cubrir las deficiencias relacionadas con el síntoma predicho por el modelo NLP

- $T$  : representa el número de tokens/palabras del texto de instrucciones de la receta

#### Métricas definidas:

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

$$Jaccard(R, F) = \frac{|R \cap F|}{|R \cup F|}$$

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

$$MissingPenalization(R, F) = \frac{|R \setminus F|}{|R|}$$

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

$$DefCov(R, H) = \frac{|R \cap H|}{|H|}$$

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 necesario

$$Effort(T) = \frac{T - T_{min}}{\max(1, T_{max} - T_{min})}$$

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:

$$0.5 * Jaccard + 0.25 * DefCov - 0.15 * MissingPenalization - 0.1 * Effort$$

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

### 6.3 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 escala (LLM) bajo el esquema de Retrieval-Augmented Generation (RAG). Este LLM actuará como juez, seleccionando 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 proviene de las predicciones de YOLO (ingredientes disponibles en la nevera del usuario) y NLP (el síntoma predicho con sus posibles deficiencias nutricionales)

- **Generación de texto:** el LLM utiliza la información recuperada para producir una respuesta final.

Para el proyecto se emplea el modelo LLaMA 3.1 8B, ejecutado mediante Ollama. Se selecciona este modelo por dos motivos principales: (1) no implica coste de uso al ejecutarse en local y (2) ofrece un equilibrio adecuado entre capacidad de razonamiento y eficiencia computacional. Los modelos LLaMA forman parte de una familia de modelos de lenguaje desarrollados por *Meta* basados en la arquitectura *Transformer*, entrenados con grandes cantidades de datos multilingües. (Meta, s.f.) La variante del modelo utilizado contiene aproximadamente 8.03 mil millones de parámetros y permite contextos de hasta 128 000 tokens.

Para poder controlar el formato, la salida, así como el comportamiento del LLM se emplean técnicas de *prompt engineering*, estructuradas en dos componentes:

- **System Prompt (instrucciones globales):** este prompt 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 con objetivos medicinales)
  - verificar siempre la viabilidad de la receta en función de los ingredientes detectados en la nevera
  - comprobar que la receta contribuye a suprir la deficiencia nutricional asociada al síntoma detectado
- **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.

Para el código de las instrucciones, véase “*Anexo E: Construcción del algoritmo de puntuación y LLM con RAG como juez*”.

En resumen, el algoritmo de puntuación hace un primer filtro reduciendo dos millones de recetas a diez recetas con mejor puntuación, y el LLM con RAG actúa como juez final aplicando razonamiento sobre esas candidatas para seleccionar las tres recetas definitivas garantizando la calidad de estas. Cabe destacar que el modelo LLM se usa con temperatura 0.1 (rango 0-1, cuanto más alto, más creatividad) para asegurar que el modelo mantenga consistencia en las respuestas, reduciendo la aleatoriedad en la generación de texto.

## 7. Productivización del sistema

En esta sección se describe el proceso de productivización del sistema, integrando los diferentes modelos desarrollados (YOLO, NLP, MarianMT para traducción, algoritmo de puntuación y modelo LLM con RAG) en una aplicación implementada con **Streamlit**, que actúa como interfaz ligera y accesible para el usuario. Esta herramienta permite ejecutar de forma sencilla todo el *pipeline* y flujo descrito en la sección 2. *Flujo de la aplicación y algoritmos utilizados*.

A continuación, se muestra visualmente cómo funcionaría la aplicación y cómo el usuario puede obtener recetas personalizadas a través de la aplicación de Streamlit. La aplicación se organiza en tres secciones principales:



**Recetas recomendadas**

JHolal Me alegra ayudarte.

Síntoma detectado: dolor de cabeza Deficiencias nutricionales posibles: falta de vitamina B6, falta de magnesio y posible déficit en ácidos grasos omega 3. Es importante mencionar que es mejor consultar con un médico si el dolor de cabeza persiste o empeora.

Ingredientes disponibles en tu nevera: ['aguacate', 'huevos', 'limón', 'manzana', 'pepino', 'platano', 'queso', 'tomate', 'yogur', 'zanahoria']

**Recetas recomendadas:**

- Katie's Favorite Salad**
  - Ingredientes necesarios: aguacate, tomate, pepino
  - Ayuda al síntoma porque el aguacate es rico en ácidos grasos omega-3 y el tomate contiene antioxidantes que pueden ayudar a reducir la inflamación.
  - Sustituciones lógicas: puedes omitir el pepino si no lo tienes en casa.
  - Instrucciones: mezcla todos los ingredientes y sirve.
- Cottage Cheese Refresher**
  - Ingredientes necesarios: yogur, tomate, zanahoria
  - Ayuda al síntoma porque el yogur es una buena fuente de calcio y la zanahoria contiene vitamina B6 que puede ayudar a reducir la inflamación.
  - Sustituciones lógicas: puedes omitir el tomate si no lo tienes en casa.
  - Instrucciones: mezcla todos los ingredientes y sirve sobre Melba toast.
- Veggie Pita**
  - Ingredientes necesarios: aguacate, pepino, queso
  - Ayuda al síntoma porque el aguacate es rico en ácidos grasos omega-3 y el pepino contiene antioxidantes que pueden ayudar a reducir la inflamación.
  - Sustituciones lógicas: puedes omitir el queso si no lo tienes en casa o si prefieres una opción vegana.
  - Instrucciones: corta una pita en dos mitades, rellénalo con aguacate, pepino y queso, y sirve.

Recuerda que es importante consultar con un médico si el dolor de cabeza persiste o empeora. Estas recetas son solo una sugerencia para ayudar a aliviar los síntomas.

**1. Descripción del estado del usuario:** se solicita al usuario que introduzca una breve descripción de su estado anímico o físico reciente. Esta información se procesará con el modelo NLP, construido en la sección 5. *Construcción del modelo NLP*, que predice el síntoma asociado y determina posibles deficiencias nutricionales. A partir de estas deficiencias, se genera también una lista de ingredientes (entre las 30 clases definidas) que podrían contribuir a suplirlas.

- 2. Subida de imagen de la nevera:** el usuario carga una fotografía de su nevera, la cual será procesada con el modelo **YOLO** construido en la sección 4. *Creación del modelo YOLO* para detectar de forma automática los ingredientes disponibles.
- 3. Recomendación personalizada:** finalmente, al hacer clic en el botón “Recomiéndame recetas”, la aplicación ejecuta el *pipeline* completo. Se procesan los *inputs* con los modelos YOLO y NLP, se carga el dataset interno de recetas y se traduce al español. Después, se aplica el sistema de puntuación definido en la sección 6. *Algoritmo de recomendación y LLM juez con RAG* para seleccionar las diez mejores opciones. Estas candidatas son evaluadas por el LLM con RAG, que, utilizando el contexto obtenido de los ingredientes y síntomas, filtra las recetas menos viables y selecciona las tres más adecuadas. Como valor añadido, el sistema proporciona al usuario no solo la recomendación final, sino también el razonamiento y la justificación detrás de cada elección.

Para más detalle, consulte el código completo en *Anexo F: Código completo de las funciones e interfaz de la aplicación*.

## 8. Conclusiones y resultados

Al finalizar este proyecto se ha conseguido construir **HealthBite**, un prototipo de aplicación inteligente capaz de recomendar recetas personalizadas en función del contexto individual del usuario. La personalización trasciende las soluciones convencionales disponibles en el mercado. La aplicación incorpora dos elementos innovadores:

- (1) La posibilidad de que el usuario describa su estado anímico o físico para inferir posibles deficiencias nutricionales. (2) La capacidad de subir una fotografía de su nevera, de la cual el sistema detecta automáticamente los ingredientes disponibles en lugar de requerir introducción manual

De esta forma, HealthBite responde a necesidades aún no cubiertas por las aplicaciones actuales, que suelen limitarse al escaneo de códigos de barras o a la recomendación de recetas genéricas enfocadas en dietas específicas (generalmente saludables) sin tener en consideración el contexto o necesidad de cada persona. Adicionalmente, la propuesta aporta un valor añadido en sostenibilidad al contribuir a la reducción del desperdicio alimenticio. Dado que el sistema identifica automáticamente los ingredientes de la nevera, puede detectar ingredientes olvidados o poco visibles en la nevera y proponer recetas que los prioricen e incorporen, mitigando uno de los principales factores de desperdicio alimenticio según los estudios. Además, introduce un enfoque preventivo en salud al vincular descripciones subjetivas con posibles deficiencias nutricionales y sugerir recetas que podrían ayudar a corregirlas.

Durante el proyecto se ha permitido validar la viabilidad de construir una aplicación inteligente que combina visión artificial (YOLO), procesamiento de lenguaje natural (NLP - BERT adaptado al español) y un sistema de recomendación apoyado en un modelo LLM con RAG para ofrecer recetas personalizadas. Los resultados experimentales muestran un rendimiento sólido en cada uno de los componentes:

**Visión artificial (modelo YOLO):** el modelo final entrenado con 100 epochs y dataset mixto (público + sintético) alcanza valores en métricas de precisión (~0.96), recall (~0.91) y mAP50-95 (~0.82), comparables a estándares de estado del arte, demostrando su capacidad para identificar correctamente los ingredientes en condiciones realistas.

**Procesamiento de lenguaje natural (modelo NLP):** El fine-tuning sobre BERT adaptado al español, entrenado con un dataset compilado de fuentes médicas y ejemplos sintéticos, mostró métricas muy altas (accuracy y F1 cercanas a 1.0). Sin embargo, este rendimiento debe interpretarse con cautela por el posible riesgo de *data leakage*. El modelo evidencia un gran potencial para identificar síntomas y asociarlos a deficiencias nutricionales.

**Sistema de recomendación integrado con LLM con RAG:** El sistema de recomendación basado en métricas definidas (similitud de Jaccard, cobertura de deficiencias, penalización por faltantes y esfuerzo requerido), demostró ser efectivo para filtrar un conjunto inicial de más de dos millones de recetas y priorizar aquellas más adecuadas al contexto del usuario. En una segunda fase, la incorporación de un modelo LLM bajo el esquema *Retrieval-Augmented Generation (RAG)* añadió una capa de razonamiento avanzado que permitió validar la viabilidad de las recetas candidatas, descartar aquellas que no cumplían las condiciones mínimas y, además, generar explicaciones claras y personalizadas. Esta combinación no solo incrementa la robustez del sistema, sino que también enriquece la experiencia del usuario al ofrecer transparencia y justificación sobre el porqué de cada recomendación.

En conjunto, los resultados confirman que la integración de estos componentes en una aplicación construida con *Streamlit* ofrece una experiencia coherente, innovadora y funcional, constituyendo un prototipo prometedor para la nutrición personalizada y la reducción del desperdicio alimenticio.

## 8.1 Limitaciones encontradas y futuras mejoras

A pesar de los logros, el proyecto también presenta limitaciones que al mismo tiempo pueden ser oportunidades para futuras mejoras:

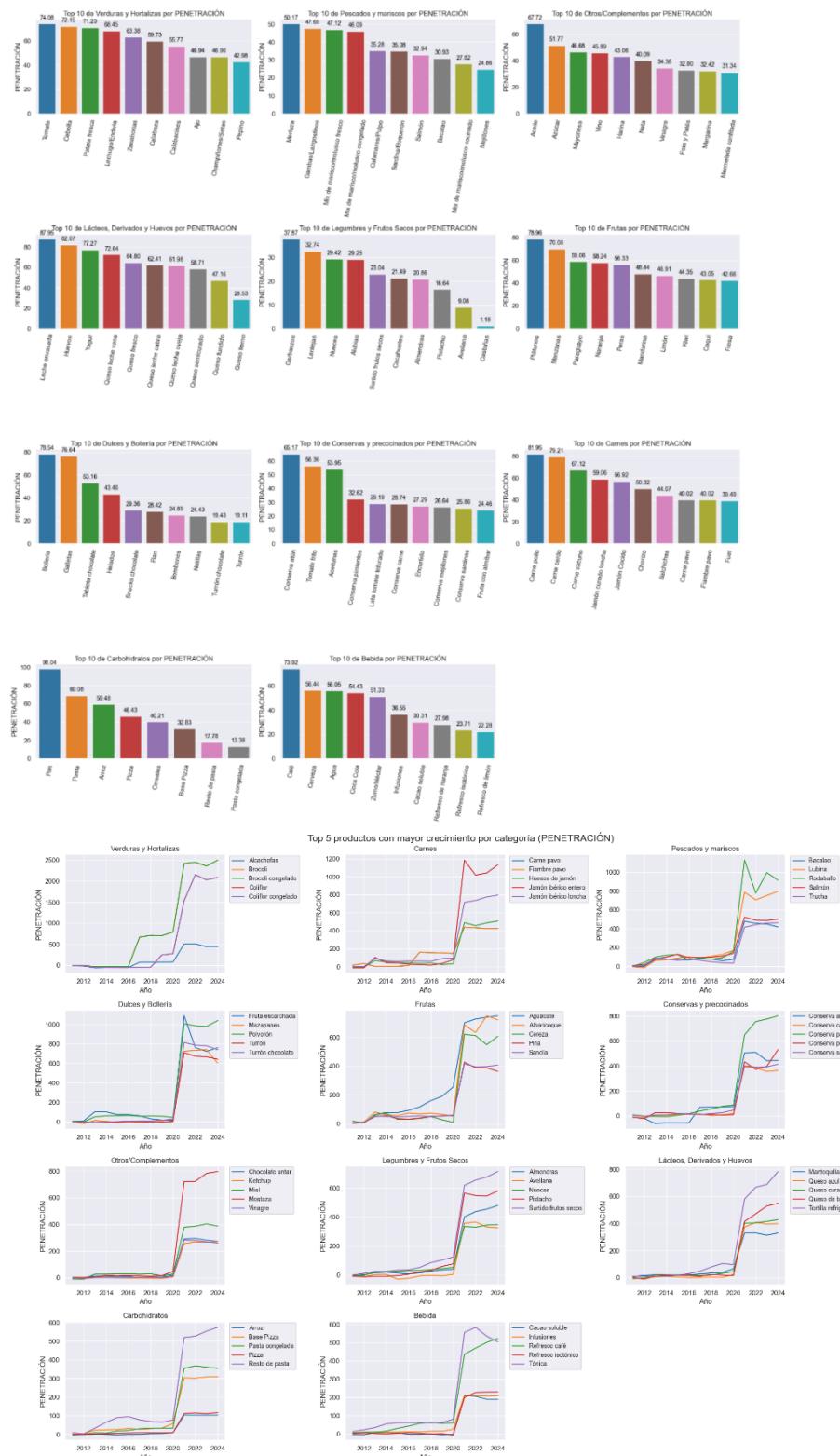
1. **Cobertura limitada de ingredientes:** Por restricciones computacionales, el modelo YOLO se entrenó únicamente con 30 clases, por lo que actualmente solo es capaz de detectar ingredientes específicos. En un entorno de producción real sería necesario ampliar considerablemente la variedad de ingredientes reconocidos para adaptarse a la diversidad real de los hogares.
2. **Contexto restringido a la nevera:** Actualmente, el sistema detecta ingredientes solo en fotografías de neveras. Una mejora sería ampliar la detección a otros contextos, como despensas o mesas de cocina, para poder abarcar más escenarios donde los usuarios pueden tener los alimentos.
3. **Necesidad de ampliar el dataset NLP:** El modelo NLP que procesa descripciones y predice síntomas está actualmente entrenado con un dataset combinado de compilación manual de fuentes médicas y generaciones artificiales sintéticas. Esto limita la variedad que pueda haber y una futura mejora sería incorporar datos reales a medida que se colecciona más información con el uso.
4. **Mejora del entorno de despliegue:** El prototipo funciona en entorno local. Una mejora fundamental sería migrarlo a la nube y desarrollar una aplicación móvil que facilite su acceso masivo.
5. **Potencial de monitoreo de patrones de salud a largo plazo con dashboards:** La aplicación presenta un gran potencial de evolución hacia la monitorización continua de la salud del usuario mediante dashboards interactivos. Cada vez que se introduce una descripción, los síntomas y deficiencias nutricionales detectados por el modelo podrían almacenarse en una base de datos SQL y actualizarse automáticamente en el panel. Esto permitiría identificar tendencias y patrones a lo largo del tiempo, ayudando al usuario a descubrir patrones antes inadvertidos. Por ejemplo, detectar la recurrencia de dolores de cabeza los martes y vincularla con hábitos como trasnochar la noche anterior o con picos de estrés laboral, aportando así un valor añadido en prevención y autogestión de la salud.

# Anexo

Todo el código está disponible en un repositorio de Github, véase: [TFM HealthBite código](#). En las secciones siguientes se presentarán diferentes gráficas, ejemplos ilustrativos y fragmentos relevantes de código. Sin embargo, no se incluirá la totalidad de los códigos en la mayoría de secciones dado que son extensos. No obstante, se proporcionarán enlaces directos a los notebooks correspondientes para consultar el desarrollo completo.

## Anexo A: Código y gráficas del análisis exploratorio de datos

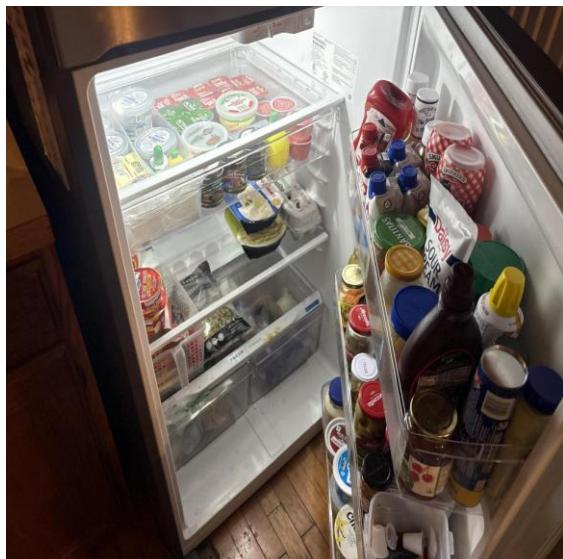
Para el código entero, véase específicamente: [Análisis de exploratorio de datos.ipynb](#)



## Anexo B: Código y fotografías ejemplo del set de datos del modelo YOLO

- Para el código entero de la construcción del dataset para el modelo de detección de objetos YOLO, véase: [Construcción de dataset de detección de objetos.ipynb](#)
- Para el código entero para el entrenamiento del modelo YOLO, véase: [Entrenamiento del modelo YOLO.ipynb](#)

### Ejemplos de imágenes usadas para el entrenamiento



### Ejemplo de fotografías y fondos de nevera descargados para la construcción de datasets sintéticos



### Ejemplo de fotografías sintéticas de ingredientes en la nevera



## Ejemplo de fotografías sintéticas poco realistas antes de delimitar las áreas posibles



## Estructura de capas y parámetros del modelo YOLO (YOLO11n)

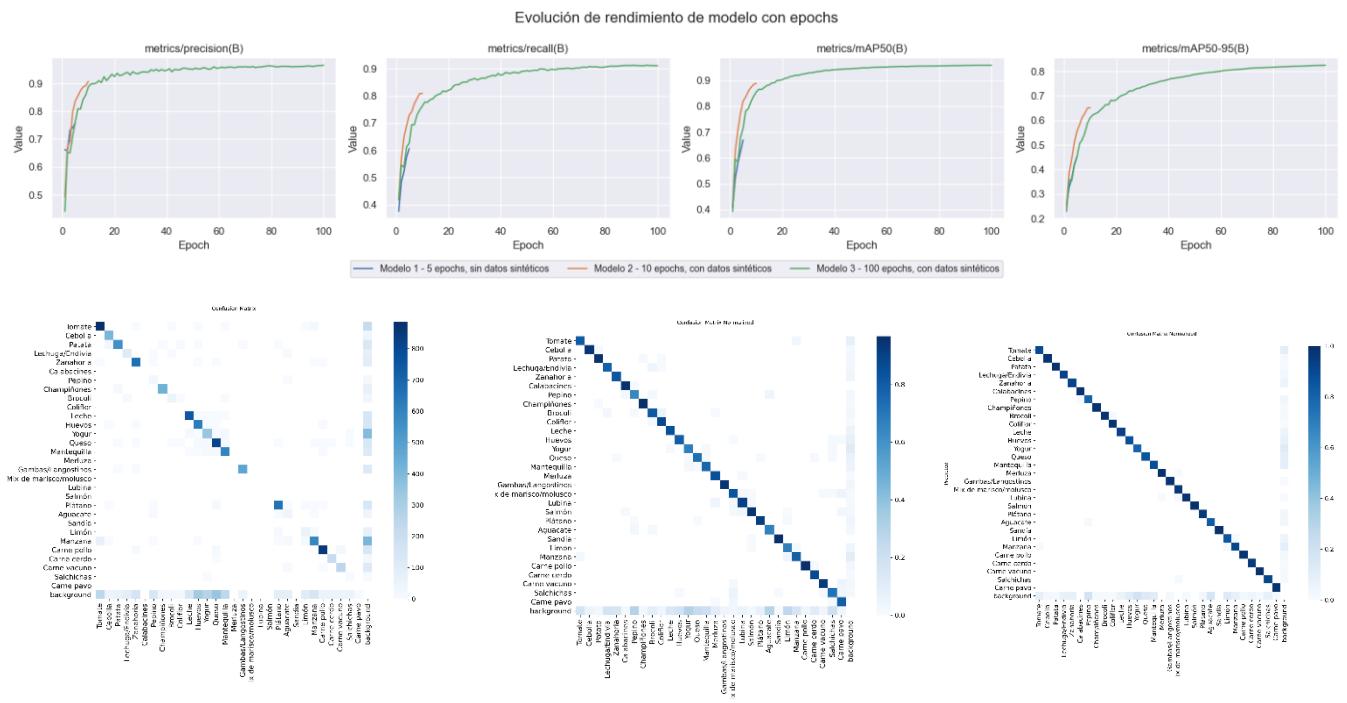
	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	6640	ultralytics.nn.modules.block.C3k2	[32, 64, 1, False, 0.25]
3	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
4	-1	1	26080	ultralytics.nn.modules.block.C3k2	[64, 128, 1, False, 0.25]
5	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
6	-1	1	87040	ultralytics.nn.modules.block.C3k2	[128, 128, 1, True]
7	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
8	-1	1	346112	ultralytics.nn.modules.block.C3k2	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	249728	ultralytics.nn.modules.block.C2PSA	[256, 256, 1]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
13	-1	1	111296	ultralytics.nn.modules.block.C3k2	[384, 128, 1, False]
14	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
15	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
16	-1	1	32096	ultralytics.nn.modules.block.C3k2	[256, 64, 1, False]
17	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
18	[-1, 13]	1	0	ultralytics.nn.modules.conv.Concat	[1]
19	-1	1	86720	ultralytics.nn.modules.block.C3k2	[192, 128, 1, False]
20	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
21	[-1, 10]	1	0	ultralytics.nn.modules.conv.Concat	[1]
22	-1	1	378880	ultralytics.nn.modules.block.C3k2	[384, 256, 1, True]
23	[16, 19, 22]	1	436522	ultralytics.nn.modules.head.Detect	[30, [64, 128, 256]]

YOLO11n summary: 181 layers, 2,595,690 parameters, 2,595,674 gradients, 6.5 GFLOPs

```
mode: train
model: yoloin.pt
data: ..\dataset\YOLO - Clean dataset\Clean dataset 2\split dataset\data.yaml
epoch: 100
time: null
patience: 10
batch: 16
imgsz: 768
save: true
save_period: -1
cache: false
device: null
workers: 8
project: null
name: train
exist_ok: false
prevalence: true
optimizer: auto
verbose: true
seed: 0
deterministic: true
single_cls: false
rect: false
cos_lr: false
close_mosaic: 10
resume: false
augment: true
fraction: 1.0
profile: false
freeze: null
multi_scale: false
overlap_mask: true
mask_ratio: 4
dropout: 0.0
val: true
split: val
save_json: false
conf_thres: 0.01
iou: 0.7
max_det: 300
half: false
dn: false
plots: true
source: null
vid_stride: 1
stream_buffer: false
```

```
flipud: 0.0
fliplr: 0.5
bgr: 0.0
mosaic: 1.0
mixup: 0.0
cutmix: 0.0
copy_paste: 0.0
copy_paste_mode: flip
auto_augment: randaugment
erasing: 0.4
cfg: null
tracker: botsort.yaml
save_dir: runs\detect\train
```

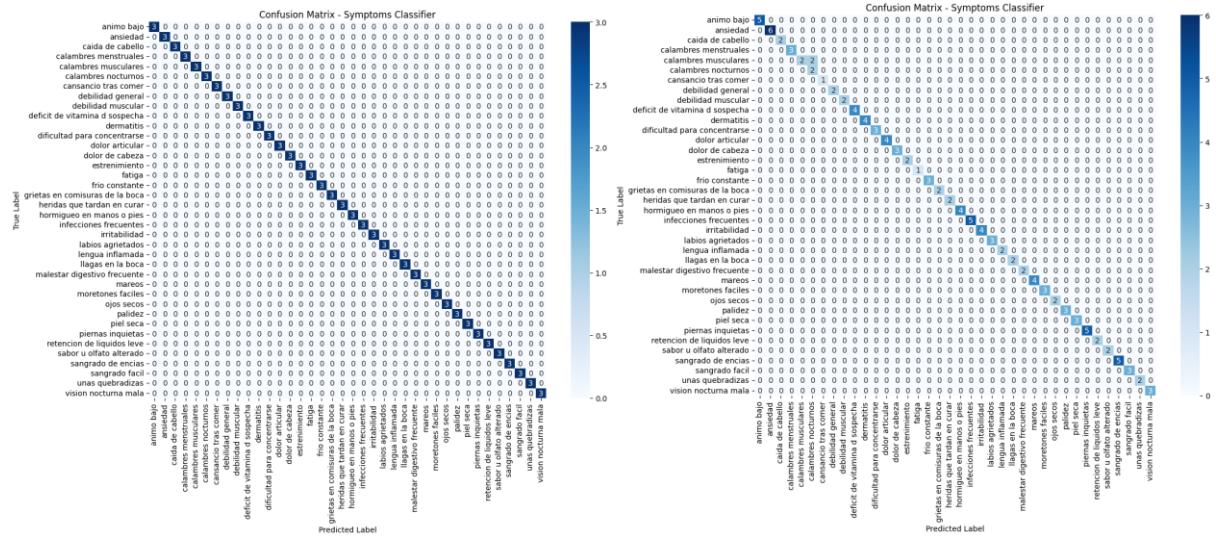
## Matrices de confusión y gráficas adicionales de evaluación del modelo



## Anexo C: Entrenamiento del modelo NLP

Para el código entero del entrenamiento del modelo NLP, véase: [Entrenamiento del modelo NLP.ipynb](#)

## Matrices de confusión de la evaluación del modelo



## Configuración de entrenamiento y métricas de evaluación

```
# 4) Métricas de evaluación
# -----
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro", zero_division=0),
        "precision_macro": precision_score(labels, preds, average="macro", zero_division=0),
        "recall_macro": recall_score(labels, preds, average="macro", zero_division=0),
    }

# 5) Configuración de entrenamiento
# -----
args = TrainingArguments(
    output_dir=r"..\models\NLP\NLP runs - stratified",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    logging_strategy="steps",
    logging_steps=1,
    num_train_epochs=10,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,
    fp16=True,
    seed=seed
)

trainer = Trainer(
    model=model_NLP,
    args=args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["validation"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=5)]
)
```

## Anexo D: Traducción del dataset de recetas

Para el código entero de la traducción del dataset de recetas, véase: [Traducción del dataset de recetas.ipynb](#)

Snippet de código de mapeo de términos inglés <> español y modelo MarianMT para traducción

```
en_spa_ingredients = {
    "Tomate": ["tomato", "roma tomato", "cherry tomato", "tomatillo"],
    "Cebolla": ["onion", "red onion", "white onion", "yellow onion", "shallot", "green onion", "spring onion", "scallion"],
    "Patata": ["potato", "new potato", "russet", "yukon gold", "baking potato"],
    "Zanahoria": ["carrot", "orange", "romaine", "butterhead", "bio", "cos", "endive", "escarole"],
    "Calabacines": ["zucchini", "courgette", "summer squash"],
    "Pepino": ["cucumber", "english cucumber", "kirby"],
    "Champiñones": ["mushroom", "button mushroom", "cremini", "portobello", "shiitake", "oyster mushroom", "chanterelle", "porcini"],
    "Brocoli": ["broccoli", "broccolini"],
    "Coliflor": ["cauliflower"],

    "Leche": ["milk", "whole milk", "skim milk", "2% milk", "evaporated milk"],
    "Huevos": ["egg", "eggs"],
    "Yogur": ["yogurt", "greek yogurt", "yoghurt"],
    "Queso": ["cheese", "cheddar", "mozzarella", "parmesan", "feta", "gouda", "goat cheese", "blue cheese", "ricotta", "cream cheese", "swiss"],
    "Mantequilla": ["butter", "unsalted butter", "salted butter", "ghee"],

    "Merluza": ["hake"],
    "Gambas/Langostinos": ["shrimp", "prawn", "prawns", "king prawn"],
    "Mix de marisco/molusco": ["seafood mix", "mixed seafood", "clams", "mussels", "oysters", "scallops", "squid", "calamari", "octopus"],
    "Lubina": ["sea bass", "seabass", "branzino", "european seabass"],
    "Salmón": ["salmon"],

    "Plátano": ["banana", "plantain"],
    "Aguacate": ["avocado"],
    "Sandía": ["watermelon"],
    "Límon": ["lemon"],
    "Manzana": ["apple", "granny smith", "gala apple", "fuji apple"],

    "Carne pollo": ["chicken", "chicken breast", "chicken thigh", "chicken leg", "rotisserie chicken", "ground chicken"],
    "Carne cerdo": ["pork", "pork loin", "pork chop", "pork shoulder", "ground pork", "bacon"],
    "Carne vacuno": ["beef", "steak", "ground beef", "silcolin", "ribeye", "chuck", "brisket"],
    "Salchichas": ["sausage", "sausages", "hot dog", "frankfurter", "chorizo", "kielbasa", "bratwurst"],
    "Carne pavo": ["turkey", "ground turkey", "turkey breast", "turkey mince"],
```

```
mt_name = "Helsinki-NLP/opus-mt-en-es"
device = "cuda" if torch.cuda.is_available() else ("mps" if torch.backends.mps.is_available() else "cpu")
print(f"Using device: {device}")

mt_tok = MarianTokenizer.from_pretrained(mt_name)
mt_model = MarianMTModel.from_pretrained(mt_name).to(device)
mt_model.eval()

def translate_batch(texts, src_max_len=256, max_new_tokens=128):
    if not texts:
        return []
    with torch.inference_mode():
        inputs = mt_tok(
            texts,
            return_tensors="pt",
            padding=True,
            truncation=True,
            max_length=src_max_len,
        ).to(device)
        outputs = mt_model.generate(
            **inputs,
            num_beams=1,
            do_sample=False,
            max_new_tokens=max_new_tokens,
            use_cache=True,
        )
    return mt_tok.batch_decode(outputs, skip_special_tokens=True)

BATCH = 96 if device == "cuda" else 24
en2es = []
total = len(unknown_en)
for i in range(0, total, BATCH):
    chunk = unknown_en[i:i+BATCH]
    trans = translate_batch(chunk)
    for src, tgt in zip(chunk, trans):
        en2es.append((src, tgt))
    print(f"Progreso: {(min(i+BATCH, total))/total} traducidos")

print("Cantidad de términos traducidos:", len(en2es))
```

## Anexo E: Construcción del algoritmo de puntuación y LLM con RAG como juez

Para el código entero del algoritmo de puntuación y LLM con RAG como juez, véase: [Algoritmo de recomendación de recetas.ipynb](#)

### Ejemplo de resultado tras la puntuación basado en reglas

	title	ingredients	directions	NER_terms_es	recipe_ingredients	jaccard score	missing penalization	DRC coverage	effort	predicted_symptom	nutritional_deficiency	ingredients_supplying_deficiency	fridge_ingredients_available	final_score
1857177	Buffalo-Style Chick'n Salad	["2 frozen BOCA Spicy Chick'n Veggie Patties", "..."]	["Cook patties as directed on pkg.", "..."]	["Huevos", "ajo", "Zanahoria", "Tomate", "Pep..."]	[huevos, ajo, zanahoria, tomate, pepino, queso]	0.500000	0.166667	0.20	0.011494	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.273851
2166456	Market-Fresh Salad	["4 small boneless skinless chicken breasts (..."]	["Heat barbecue to medium-high heat.", "..."]	["Came pollo", "Tomate", "Lechuga/Endivia", "..."]	[carne pollo, tomate, lechuga/endivia, zanahor...	0.454545	0.285714	0.25	0.020764	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.244839
1248534	Mexican Puffs	["1 None avocado, diced", "..."]	["Preheat the oven to 425°u00b0F/215°u00b0C.", "..."]	["Aguacate", "Tomate", "Queso", "Pastelera", "..."]	[aguacate, tomate, queso, pastelera, huevos, ...]	0.454545	0.285714	0.25	0.039303	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.242985
1266234	Slimmer'S Lunch	["1 None hard-boiled egg, halved", "..."]	["Arrange all ingredients in an airtight lunch..."]	["Huevos", "Queso", "Tomate", "ajo", "Zanahor..."]	[huevos, queso, tomate, ajo, zanahoria, remol...	0.454545	0.285714	0.20	0.004079	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.234008
1769104	Mediterranean English Muffin Sandwich	["1/2 small carrot, grated", "..."]	["Mix the carrots, cucumbers and tomatoes toge..."]	["Zanahoria", "Pepino", "Tomate", "Humus", "..."]	[zanahoria, pepino, tomate, humus, queso, agu...	0.454545	0.285714	0.20	0.019281	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.232488
666717	Turkey Cobb Salad	["3 oz. Backyard Grill turkey breast, sliced", "..."]	["Layer lettuce, tomatoes, cucumber, egg, and ..."]	["Carne pavo", "Tomate", "Queso", "Pepino", "..."]	[carne pavo, tomate, queso, pepino, huevos]	0.400000	0.200000	0.25	0.007045	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.231796
2102536	Vegie Salibatbs! (- Tasty Dish-)	["1/2 picking cucumber, cut in half lengthwise, ..."]	["BOAT: Fill the little sailboats with the ex..."]	["Pepino", "Tomate", "Pimientos amarillos", "..."]	[pepino, tomate, pimientos amarillos, huevos ...]	0.454545	0.285714	0.20	0.029663	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.231449
1086938	Ww 5 Points - Shrimp Louis Salad	["1/4 cup cottage cheese", "..."]	["In blender combine cottage cheese, tomato ju..."]	["Queso", "Tomate", "Huevos", "mostaza", "Pep..."]	[queso, tomate, huevos, mostaza, pepino, gamba...	0.416667	0.375000	0.30	0.025213	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.224562
1684146	Katie's Favorite Salad	["1/2 medium diced tomato", "..."]	["Toss ingredients and serve."]	["Tomate", "Pepino", "Aguacate", "Nueces", "..."]	[tomate, pepino, aguacate, nueces, queso]	0.400000	0.200000	0.20	0.001483	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.219852
2069448	Veggie Pita	["1 large wheat pita bread", "..."]	["Cut tops off pita to form a pocket.", "..."]	["Pan", "Aguacate", "Pepino", "Queso", "Tomate"]	[pan, aguacate, pepino, queso, tomate]	0.400000	0.200000	0.20	0.005933	dolor de cabeza	magnesio; riboflava(b2); potasio	[aguacate, platano, patata, tomate, calabacine...]	[aguacate, huevos, limon, manzana, pepino, pla...	0.219407

### Propmts utilizados para el LLM

```
system_prompt_text = (
    f"""
    Eres un asistente culinario y nutricional que se encargará de escoger las {top_n_final} mejores recetas para un usuario en base a los inputs del usuario y el contexto que te voy a dar.

    El input del usuario son los siguientes:
    Entrada del usuario: {user_text}
    Síntoma detectado: {detected_symptom}
    Ingredientes disponibles: {available_ingredients}

    El Contexto con las 10 recetas candidatas {{context_text}}.

    Tarea: elige las {top_n_final} MEJORES recetas usando SOLO el contexto.

    Criterios:
    - Usa EXCLUSIVAMENTE la información del contexto proporcionado, no inventes nada
    - Tienes que verificar si la receta es verdad es para una comida. Es decir, descarta recetas de cremas, mascarillas y otras cosas que usan ingredientes de comida pero realmente no son para la nutrición
    - Tienes que verificar que la receta es realista para hacer, es decir que el usuario tiene bastantes ingredientes en la nevera de los requeridos por la receta
    - Tienes que verificar si la receta de verdad cubriría con la posible deficiencia nutricional asociada al síntoma detectado
    - No te olvides de dar las instrucciones de la receta {recipe_instructions}
    """
)
```

```
user_prompt_text = (
    f"""
    Acuérdate que no puedes usar ninguna información que no existe o inventártelo.
    El texto que me tienes devolver tiene que seguir el siguiente formato:
    - Primero, me tienes que hablar directamente, no uses tercera persona
    - Segundo, dime que síntoma me detectas en base a mi texto
    y que deficiencias nutricionales puede que padezca en base a {deficiency_list}. No te inventes ninguna nueva deficiencia, cita exclusivamente de {deficiency_list} (Asegúrate de mencionar que siempre es mejor visitar el médico si es grave)
    - Tercero, dime los ingredientes detectados en mi nevera ({available_ingredients})
    - En base a esta información, recomiéndame recetas:

    Para cada receta elegida, tienes que explicarme:
    - El nombre de la receta {recipe_title}
    - Los ingredientes necesarios de la receta {recipe_ingredients} (traducidos al español)
    - Cita qué alimentos o ingredientes cubren las deficiencias {deficiency_covered_by_ingredients} asociadas a {detected_symptom}. No te inventes nada
    - Menciona algunas sustituciones lógicas. Solo si el usuario no tiene algún ingrediente de la receta pero si lo tiene en la nevera. No hace falta mencionar sustituciones de ingredientes comunes que todo el mundo tendría como agua, sal, aceite, etc.
    - Detallar las instrucciones y los pasos de la receta (información disponible en {recipe_instructions})"""
)
```

## Anexo F: Código completo de las funciones e interfaz de la aplicación

En esta sección se detallará el código entero utilizado para la aplicación funcional en Streamlit, véase el código en: [Código aplicación final](#)

Por otro lado, para el código de las funciones llamadas de modelo y procesamiento, veáse: [Código de funciones clave](#)

```

import os, json, uuid
import io
from datetime import datetime
import numpy as np
import pandas as pd
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from ultralytics import YOLO
import unicodedata
import re
import requests
import ast
from sklearn.preprocessing import MultitlabelBinarizer
from scipy import sparse
from typing import List, Tuple, Optional
import streamlit as st
from app_key.functions import *
from pathlib import Path

from app_key.functions import (
    NLP_YOLO_predictor_function,
    compute_recipe_scores,
    rank_recipes,
    run_rag_over_top10_human_output,
    safe_to_list,
)
)

st.image(r'app_images\HealthBite_banner_image.png')
st.markdown("<h1 style='text-align: center;'>HealthBite</h1>", unsafe_allow_html=True)
st.markdown("<p style='text-align: left;'>Un recomendador inteligente de recetas que detecta ingredientes en tu nevera y también intenta ayudarte a sentir mejor a través de una nutrición adecuada</p>", unsafe_allow_html=True)
st.divider()

# Sección para describir el estado físico/animado
st.subheader("1) Describe cómo te sientes últimamente")
user_text = st.text_area(
    "¿Cómo te has sentido últimamente?",
    placeholder="Por ejemplo: llevo días que me duele la cabeza constantemente",
)
st.divider()

# Sección para describir el estado físico/animado
st.subheader("1) Describe cómo te sientes últimamente")
user_text = st.text_area(
    "¿Cómo te has sentido últimamente?",
    placeholder="Por ejemplo: llevo días que me duele la cabeza constantemente",
)
st.divider()

# Sección para la foto de nevera
st.subheader("2) Sube una foto de tu nevera")
photo = st.file_uploader("Foto de tu nevera", type=["jpg", "jpeg", "png", "webp"])
st.divider()

# Botón
go = st.button("👉 Recomiéndame recetas", type="primary", use_container_width=True)
if go:
    with st.spinner("Procesando..."):

        # Guardando la foto cuando el usuario sube foto
        image_path = None
        if photo is not None:
            tmp_dir = Path(".tmp_inputs")
            tmp_dir.mkdir(parents=True, exist_ok=True)
            image_path = str(tmp_dir / f'fridge_{photo.name}')
            with open(image_path, "wb") as f:
                f.write(photo.read())

        # Se carga el dataset de recetas traducidas
        df_recipes = pd.read_csv(r'..\dataset\Recipes dataset\recipes_dataset_translated_ingredients.csv')

        # Paso 1+2: Predictor de síntoma y detector de ingredientes disponibles en la nevera
        pred_df = NLP_YOLO_predictor_function(user_text, image_path=image_path)
        pred_df = pred_df[['run_id', 'timestamp', 'predicted_symptom', 'fridge_ingredients_available', 'nutritional_deficiency', 'ingredients_supplying_deficiency']]
        pred_df = pred_df.rename(columns= {'timestamp': 'marca de tiempo', 'predicted_symptom': 'síntoma predecido',
                                            'fridge_ingredients_available': 'ingredientes en la nevera',
                                            'nutritional_deficiency': 'posibles deficiencias nutricionales',
                                            'ingredients_supplying_deficiency': 'ingredientes nutricionales útiles'})

```

```

# Sección para describir el estado físico/animado
st.subheader("1) Describe cómo te sientes últimamente")
user_text = st.text_area(
    "¿Cómo te has sentido últimamente?",
    placeholder="Por ejemplo: llevo días que me duele la cabeza constantemente",
)
st.divider()

# Sección para la foto de nevera
st.subheader("2) Sube una foto de tu nevera")
photo = st.file_uploader("Foto de tu nevera", type=["jpg", "jpeg", "png", "webp"])
st.divider()

# Botón
go = st.button("👉 Recomiéndame recetas", type="primary", use_container_width=True)
if go:
    with st.spinner("Procesando..."):

        # Guardando la foto cuando el usuario sube foto
        image_path = None
        if photo is not None:
            tmp_dir = Path(".tmp_inputs")
            tmp_dir.mkdir(parents=True, exist_ok=True)
            image_path = str(tmp_dir / f'fridge_{photo.name}')
            with open(image_path, "wb") as f:
                f.write(photo.read())

        # Se carga el dataset de recetas traducidas
        df_recipes = pd.read_csv(r'..\dataset\Recipes dataset\recipes_dataset_translated_ingredients.csv')

        # Paso 1+2: Predictor de síntoma y detector de ingredientes disponibles en la nevera
        pred_df = NLP_YOLO_predictor_function(user_text, image_path=image_path)
        pred_df = pred_df[['run_id', 'timestamp', 'predicted_symptom', 'fridge_ingredients_available', 'nutritional_deficiency', 'ingredients_supplying_deficiency']]
        pred_df = pred_df.rename(columns= {'timestamp': 'marca de tiempo', 'predicted_symptom': 'síntoma predecido',
                                            'fridge_ingredients_available': 'ingredientes en la nevera',
                                            'nutritional_deficiency': 'posibles deficiencias nutricionales',
                                            'ingredients_supplying_deficiency': 'ingredientes nutricionales útiles'})

```

```

st.divider()

st.subheader("Resultado de las predicciones")
st.markdown("<p style='text-align: left;'>En base a la foto que has subido y tu descripción animado/físico, detecto:</p>", unsafe_allow_html=True)
st.dataframe(pred_df, use_container_width=True)
st.markdown("<p style='text-align: left;'>Dado estos resultados, te recomendaré recetas que encajen con tus ingredientes disponibles y que puedan ayudar con posibles deficiencias nutricionales</p>", unsafe_allow_html=True)

row = pred_df.iloc[0]
fridge_list = row['ingredientes en la nevera']
deficiency_list = row['ingredientes nutricionales útiles']

# Paso 3: Puntuación y ranking de recetas
scored = compute_recipe_scores(
    df_recipes,
    fridge_list,
    deficiency_ingredients_list=deficiency_list
)

scored = scored.merge(pred_df, how='cross')
top10 = rank_recipes(scored, top_n=10)

# st.subheader("Top 10 recetas candidatas")
# st.dataframe(
#     top10[['title', 'jaccard_score', 'DRC_coverage', 'missing_penalization', 'effort', 'final_score']],
#     use_container_width=True, height=350
# )

# Paso 4: RAG LLM como juez
final_text = run_rag_over_top10_human_output(
    top10_df=top10,
    user_text=user_text or '',
    model_name="llama3.1:8b",
    top_n_final=3
)

st.subheader("Recetas recomendadas")
st.markdown(final_text)

st.success("Recetas listas!")

```

## Bibliografía

- Chen, Q., & Wang, J. (2022). Obtenido de <https://arxiv.org/pdf/2211.03594v1>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. N. (2018). *Google*. Obtenido de <https://research.google/pubs/bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding/>
- Eurostat. (September de 2024). *Eurostat*. Obtenido de [https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Food\\_waste\\_and\\_food\\_waste\\_prevention\\_-\\_estimates](https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Food_waste_and_food_waste_prevention_-_estimates)
- Heidenstrøm, N., & Hebrok, M. (May de 2020). *ScienceDirect*. Obtenido de <https://www.sciencedirect.com/science/article/abs/pii/S0195666321002282>
- Jocher, G., & Qiu, J. (2024). *Ultralytics YOLO11*. Obtenido de <https://docs.ultralytics.com/models/yolo11/#citations-and-acknowledgements>
- Junczys-Dowmunt, M. (April de 2018). Obtenido de <https://arxiv.org/abs/1804.00344>
- Longqiang Zhao, Min, S., Wang, X., & Yu, X. (November de 2024). *ScienceDirect*. Obtenido de <https://www.sciencedirect.com/science/article/abs/pii/S0921344925001533>
- Meta. (s.f.). *Llama*. Obtenido de <https://www.llama.com/>
- National Institutes of Health*. (s.f.). Obtenido de [https://ods.od.nih.gov/factsheets/list-all/?utm\\_source](https://ods.od.nih.gov/factsheets/list-all/?utm_source)
- Redmon, J. (2015). *Cornell University*. Obtenido de <https://arxiv.org/abs/1506.02640>
- Yale Medicine*. (s.f.). Obtenido de <https://www.yalemedicine.org/conditions>