UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Sistema de Question Answering sobre Pliegos de Cláusulas Administrativas Particulares

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Antohi Calapaqui, Marc

Tutor/a: López Rodríguez, Damián

Cotutor/a externo: Marques Bailon, Ion

CURSO ACADÉMICO: 2023/2024

Resumen

El principal objetivo de este Trabajo Final de Grado es el desarrollo de un sistema de *Question Answering* orientado a Pliegos de Cláusulas Administrativas Particulares. Dados unos archivos que se desean analizar, el programa será capaz de extraer la información relevante: puede incluir tablas, imágenes y fórmulas matemáticas. Los campos extraídos pueden ser predefinidos, como el objeto del contrato, la duración del contrato, el presupuesto base de licitación, etc. y también pueden ser especificados por el usuario, en caso de tener alguna pregunta sobre una licitación en concreto. Para ello se desarrollará una interfaz simple con un apartado que permita subir el archivo deseado y otro apartado para que se puedan hacer preguntas específicas en caso de que el programa no incluya suficiente información en la primera ejecución. Con la finalidad de conseguir los mejores resultados posibles, se probarán varios *LLM* que permitan llevar a cabo esta tarea. Aparte de las respuestas generadas, se tendrán en cuenta también los tiempos de ejecución, los costes de cada modelo y se realizará una comparativa.

Palabras clave: Question Answering, LLM, Licitaciones, OCR, Aprendizaje Profundo, Procesamiento del Lenguaje Natural, Extracción de tablas, Extracción de fórmulas, detección de objetos, YOLO

Agradecimientos

Me gustaría agradecer en primer lugar a Sciling, la empresa que me ha dado la oportunidad de hacer las prácticas con ellos, y que me ha proporcionado los recursos necesarios para llevar a cabo este trabajo final de grado.

En segundo lugar, agradecer a mi tutor Damián López, por haber estado atento con mi progreso en las prácticas universitarias y en este proyecto.

Por último, quería agradecer a mi madre por haber insistido tanto en este TFG y hacerme ver que era el último paso para acabar mi etapa en la universidad.

Tabla de contenidos

Índice de figuras

Capítulo 1 - Introducción

1.1 Contexto

Una licitación pública es el procedimiento administrativo mediante el cual un organismo del sector público solicita ofertas de empresas privadas para llevar a cabo un proyecto. Se utiliza para la adquisición de suministros, realización de servicios o ejecución de obras (construcciones).

La entidad pública prepara los pliegos de cláusulas, que describen con detalle las especificaciones técnicas, administrativas y económicas del proyecto. Los proveedores interesados en participar, que pueden ser autónomos, pymes o grandes empresas, deben presentar propuestas que cumplan con dichas condiciones.

Finalmente, con todas las propuestas recibidas, la parte contratante las evaluará mediante una serie de parámetros establecidos en los pliegos. El grado de cumplimiento de cada parámetro se evalúa mediante la suma de puntos, por lo tanto el proyecto se adjudicará a la empresa que más puntos haya sumado.

Un ejemplo podría ser la contratación del servicio de asesoramiento y gestión laboral para la Fundación de la Comunitat Valenciana Centro de Estudios Ambientales del Mediterráneo. Se prepara el pliego de condiciones con todas las características del servicio y las empresas interesadas analizan el documento para posteriormente poder elaborar una propuesta.

1.2 Motivación

El análisis de estos documentos puede ser extremadamente costoso, ya que pueden superar las 100 páginas. Las personas encargadas de esta tarea pueden tardar varias horas en leer y obtener la información relevante, por lo que el propósito de este proyecto es simplificar el procedimiento.

Hay algunos apartados de los pliegos que tienen más importancia que otros, como pueden ser el objeto del contrato, la localización, el presupuesto, la fórmula de revisión, entre otros. Para que una empresa pueda presentar su oferta, primero debe asegurarse de que cumple con esos requisitos.

Para ello se desarrollará un sistema de *Question Answering* para Pliegos de Cláusulas Administrativas Particulares, que dados unos archivos PDF del pliego, devolverá un informe con la información más relevante acerca de los apartados mencionados anteriormente. Así se podrá decidir de forma más rápida si un proyecto tiene rentabilidad para la empresa o no.

Se ha elegido este tipo de sistema porque se basa en recibir una pregunta en lenguaje natural, por ejemplo ‘Cuál es el objeto del contrato?’, que será analizada para comprender su significado, y luego se buscará una posible respuesta dentro de los documentos

proporcionados. Esta será procesada por un modelo grande de lenguaje (*LLM*) para poder generar una respuesta coherente y comprensible para el usuario.

Un proyecto así es una forma interesante de adentrarse en el mundo del procesamiento de lenguaje natural, un campo de la inteligencia artificial que está en constante evolución. Aparte de familiarizarse con algoritmos y técnicas de PLN, como el análisis sintáctico, análisis semántico, extracción de información y generación de respuestas, también se trabajarán las habilidades de *frontend* y *backend*, ya que se desarrollará una página web simple para el usuario.

1.3 Objetivos

1.3.1 Objetivos del sistema

El objetivo principal del sistema es automatizar el proceso de analizar los documentos y generar un informe que recoja información sobre varios campos predefinidos (objeto del contrato, presupuestos, etc). Las respuestas que genera deben tener la misma coherencia y precisión que las respuestas que daría una persona.

A continuación se indica un desglose de los subobjetivos:

- Desarrollar una página web
    - Para que un usuario pueda usar la herramienta sin experiencia previa, se diseñará una web sencilla, intuitiva y visualmente atractiva.
    - Permitirá subir los documentos de los pliegos y anexos correspondientes en formato PDF.
    - Se generará un informe que responda a preguntas predefinidas, con la información de los archivos proporcionados por el usuario
    - Permitirá hacer preguntas específicas al final del proceso, en caso de que el usuario quiera más información acerca de un apartado que no aparece en el informe inicial.
    - Se podrá descargar dicho informe en formato PDF.
    - Existirá un método de autentificación para el usuario, para que pueda consultar su historial de informes.

- Extraer información
    - Se extraerá todo el contenido de los archivos PDF subidos.

- Extraer fórmulas matemáticas
    - Los pliegos pueden incluir fórmulas matemáticas complejas, con líneas de fracción y potencias.
    - Las librerías para procesar PDF no son capaces de extraer fórmulas con esos formatos, por lo que hay que buscar una solución *ad-hoc*

- Extraer tablas
    - Se pueden encontrar tablas en los pliegos o en los anexos.
    - Muchas librerías para procesar PDF no son capaces de extraer tablas de forma correcta. Hay que buscar una solución que no aumente el tiempo de computación.

- Responder de forma coherente
    - El informe debe incluir la información extraída de los documentos pero se tiene que redactar de forma coherente.
    - Se usará un modelo grande de lenguaje (*LLM*) capaz de responder en castellano.

1.3.2 Objetivos económicos y computacionales

- Usar tecnologías con la mejor relación calidad/precio
- El sistema debe tardar un máximo de 20 minutos en procesar una licitación y devolver un informe con las preguntas del usuario y sus respuestas

1.4 Estructura de la Memoria del TFG

En el apartado anterior se han presentado los objetivos del sistema que se pretende implementar. La siguiente sección se centra en examinar el estado del arte de las tecnologías que se utilizarán para cumplir con ellos. Se presentarán de forma muy detallada, con el propósito de revelar las diversas arquitecturas que existen de cada tecnología y sus potenciales.

Para extraer la información de los documentos PDF de los pliegos se usará un sistema de búsqueda y recuperación de información. Dada una consulta del usuario, el sistema será capaz de identificar los pasajes del texto que más probabilidades tienen de contener la respuesta. Se explicará el proceso que se lleva a cabo, las medidas que se usan para calcular el rendimiento y los diferentes modelos que existen para esta tarea. Finalmente se explicarán dos formas de transformar la información para que esta sea más fácil de recuperar.

Por otra parte se explicará en qué consiste la detección de objetos, ya que será necesaria para poder identificar las fórmulas matemáticas complejas que aparecen en los criterios de puntuación de los pliegos. Se introducirá con la explicación de las redes neuronales convolucionales, que están especialmente diseñadas para procesar y analizar datos visuales. Luego se detallará el proceso de entrenamiento del modelo, los distintos métodos que existen para medir su rendimiento y finalmente se presentará la arquitectura *YOLO*.

Asimismo, se encuentra un apartado para las redes neuronales que pueden ser entrenadas para tareas de *Question-Answering*, que son esenciales para este proyecto. Se inicia por las redes neuronales recurrentes, con sus variantes *LSTM* y *GRU*, y se concluye con la arquitectura de los *Transformers*, en los que están basadas las familias de los modelos grandes de lenguaje *BERT* y *GPT*.

En el apartado de desarrollo de solución se aplicará la teoría explicada en el apartado del estado del arte, para desarrollar un sistema que cumpla con los objetivos iniciales. Se detallarán cada uno de los procedimientos seguidos con el fin de asegurar la correcta funcionalidad de la herramienta. Se han estudiado diversas alternativas para cada problema que ha ido surgiendo durante el desarrollo, por lo que hay apartados en los que se

mencionan las diferentes soluciones que se han probado y se explican los pros y contras de cada una.

Finalmente, se exponen los resultados que se obtienen procesando una licitación y se da una conclusión personal del trabajo.

Capítulo 2 - Estado del Arte

El estado del arte de los sistemas de *Question Answering* está evolucionando constantemente debido a los avances en la búsqueda y recuperación de información, en el procesamiento de lenguaje natural (NLP) y el aprendizaje automático.

A continuación se explicará el contexto teórico de cada una de las tecnologías utilizadas para llevar a cabo el proyecto. Se incluirán algunos ejemplos para aclarar conceptos, pero la aplicación de esta teoría en el sistema se explicará en el apartado de Desarrollo de solución.

2.1 Búsqueda y recuperación de información

2.1.1 Introducción

La búsqueda y recuperación de información [22] es la ciencia de la búsqueda de información en documentos electrónicos y cualquier tipo de colección documental digital. Tiene como objetivo la recuperación de textos, imágenes, sonido o datos de otras características, de manera pertinente y relevante para una consulta dada.

Historia
La popularización de las computadoras como medios de búsqueda de información se originó debido a un artículo titulado '*As We May Think*' de Vannevar Bush [2]. Los sistemas de recuperación a larga escala, como *Lockheed*, comenzaron a usarse en 1970.

En 1992, el Departamento de Defensa de los Estados Unidos patrocinó la Conferencia de Recuperación de Texto (TREC) [6]. Este suministró los recursos necesarios para evaluar metodologías de recuperación de texto en largas colecciones. Los motores de búsqueda han aumentado significativamente la demanda de sistemas de recuperación con una mayor capacidad.

Proceso
Un proceso de recuperación de información inicia cuando un usuario realiza una consulta al sistema. Esta no solo puede identificar un objeto dentro de la colección, sino que también puede identificar a varios objetos, pero con diferentes grados de relevancia.

Para ello se procede a la elaboración de un *ranking* con el fin de determinar la adecuación de cada objeto a la consulta. Los que tienen mayor puntuación se muestran al usuario y el proceso puede contar con otras iteraciones si este desea mejorar su consulta.

2.1.2 Medidas de rendimiento y corrección

Existen diversas medidas que permiten evaluar el rendimiento de este tipo de sistemas [15]. A continuación se presentan algunas de ellas, asumiendo que existe una colección de documentos y una consulta, y se sabe si un documento es relevante o no para dicha consulta.

- La **precisión** es la proporción de documentos relevantes dentro de todos los recuperados para la consulta del usuario.

  $$\text{Precisión} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ recuperados\}|}$$

- La **exhaustividad (*recall*)** es la fracción de documentos relevantes recuperados dentro de todos los documentos relevantes (recuperados o no)

  $$\text{Exhaustividad} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ relevantes\}|}$$

- La **proposición de fallo** (*fall-out)*, evalúa la proporción de documentos no relevantes que son recuperados, fuera de todos los documentos no relevantes disponibles.

  $$\text{Fall-out} = \frac{|\{documentos\ no\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ no\ relevantes\}|}$$

- La **medida F** es un balance de la precisión y la exhaustividad:

  $$F = \frac{2 \cdot Precisión \cdot Exhaustividad}{(Precisión + Exhaustividad)}$$

La precisión y la exhaustividad son métricas que se basan en todos los documentos retornados por el sistema dada una consulta del usuario. Para los que hacen ranking a los documentos también es conveniente considerar el orden en que los documentos son retornados. Si se calcula la precisión y la exhaustividad en cada posición de la secuencia de documentos con ranking, se puede mostrar gráficamente la curva precisión-exhaustividad, definiendo la precisión *p(r)* como una función de la exhaustividad *r*.

- La **Precisión Promedio** computa el promedio de los valores de *p(r)* sobre la integral desde *r = 0* hasta *r = 1***.**

  $$\text{AveP} = \int_0^1 p(r)\, dr$$

  Esta integral es reemplazada en la práctica por una suma finita sobre todas las posiciones en la secuencia de documentos con ranking:

$$\text{AveP} = \sum_{k=1}^{N} P(k)\Delta r(k)$$

donde *k* es el ranking en la secuencia de documentos recuperados, *n* es el número de documentos recuperados, *P(k)* es la precisión del corte en la posición *k* de la lista y *r(k)* es el cambio en el recobrado de los elementos *k-1* hasta *k*.

- La **media de la precisión promedio** (*MAP*) es el promedio de las puntuaciones medias de precisión para cada consulta.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

donde **Q** es el número de consultas evaluadas.

2.1.3 Tipos de Modelos

Con el fin de recuperar los documentos relevantes mediante técnicas de recuperación de información, estos se transforman a una representación lógica de los mismos. Cada estrategia de recuperación contiene un modelo específico para sus objetivos de representación de los documentos, y se clasifican en función de dos dimensiones: la base matemática y las propiedades de los modelos.

Base matemática:

- Modelos basados en Teoría de Conjuntos: Los documentos se representan como un conjunto de palabras o frases.
- Modelos Algebraicos: En estos modelos los documentos y las consultas se representan como vectores. La similitud entre un documento y una consulta se representa por un escalar.
- Modelos Probabilísticos: Tratan el proceso de recuperación de documentos como una inferencia probabilística. Las similitudes son calculadas como las probabilidades de que un documento sea relevante dada una consulta.

Propiedades de los Modelos:

- Modelos sin interdependencia entre términos: Interpretan los términos como independientes.
- Modelos con interdependencia entre términos: Son capaces de representar las interdependencias entre términos.

2.1.4 Ejemplos - *One hot encoding & Word embeddings*

A continuación se explicarán dos formas de representar las palabras de un texto, osea la información, como vectores:

- Un ejemplo de representación vectorial de las palabras es la ***one hot encoding***. Se asigna una dimensión a cada palabra del vocabulario. Cada palabra del vocabulario se presenta como un vector binario con todos sus valores a cero, excepto el índice de la misma. Por ejemplo:

  <<El gato se ha sentado en el sofá>>

  tendría la siguiente codificación:

  El: [1 0 0 0 0 0 0 0]
  gato: [0 1 0 0 0 0 0 0]
  se: [0 0 1 0 0 0 0 0]
  ha: [0 0 0 1 0 0 0 0]
  sentado: [0 0 0 0 1 0 0 0]
  en: [0 0 0 0 0 1 0 0]
  el: [0 0 0 0 0 0 1 0]
  sofá: [0 0 0 0 0 0 0 1]

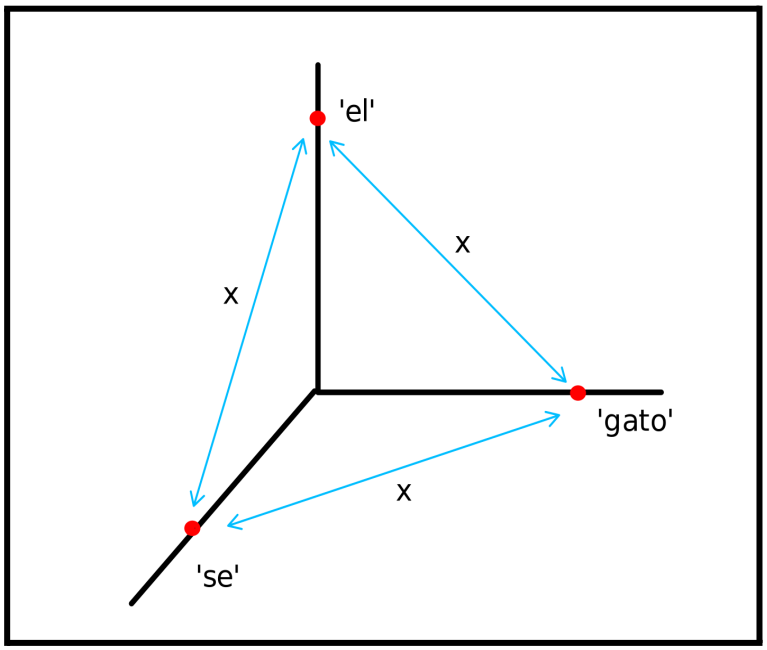


Figura 2.1: One hot encoding de las palabras: ‘el’,’gato’ y ’se’

El principio es asignar un índice a cada palabra del vocabulario. Este se representa en forma de diccionario :

vocabulario: {‘el’: 0, ‘gato’: 1, ‘se’:2, ...}

En este contexto, el vocabulario se compone de las ocho palabras singulares de nuestro *corpus*. El procedimiento consiste en representar la palabra del vocabulario como un vector de dimensión 8 que tiene todos sus valores a cero excepto el índice de la palabra.

One-hot(‘sentado’)= [0 0 0 0 1 0 0 0]

Como se puede observar en la figura 2.1, todas las palabras tienen la misma distancia (*x)* y similitud. Por tanto, la codificación one hot solo proporciona información según la cual una palabra es diferente de otra. Se puede constatar que las palabras ‘El’ y ‘el’ tienen una codificación diferente (se consideran palabras distintas), lo que puede considerarse una desventaja de este modo de representación de palabras.

- Por otra parte, el **word embedding** permite capturar el contexto, la similitud semántica y sintáctica (género, sinónimos, etc.) de una palabra reduciendo el tamaño. Por ejemplo, se podría esperar que las palabras ‘perro’ y ‘gato’ estuvieran representadas por vectores relativamente cercanos en el espacio vectorial en el que se definen.

  Queremos que el modelo elija las características más relevantes para representar la palabra. Por ejemplo, la característica ‘ser vivo’ podría ser útil para diferenciar ‘perro’ y ‘ordenador’, y para acercar ‘perro’ y ‘gato’. Ver figura 2.2.

  Los métodos para generar este tipo de representaciones son: las redes neuronales, la reducción de dimensionalidad con matrices de co-ocurrencia de palabras y los modelos probabilísticos.

  Entre el software para entrenar y utilizar *word embedding* se encuentra ‘*word2vec’* (desarrollado por Google), ‘*GloVe’* (desarrollado por el equipo de Stanford NLP), *TensorFlow, Keras,* entre otros. Algunas de sus características es que pueden manejar palabras desconocidas utilizando sub-palabras (n-gramas de caracteres), lo que mejora la representación de palabras raras. También permiten entrenar modelos de *embeddings* personalizados y utilizar *embeddings* pre-entrenados como parte de redes neuronales más grandes.

  Estas técnicas demostraron aumentar el rendimiento de tareas en el procesamiento del lenguaje natural (NLP) como la clasificación de texto, reconocimiento de entidades nombradas, análisis de sentimiento y muchas otras.

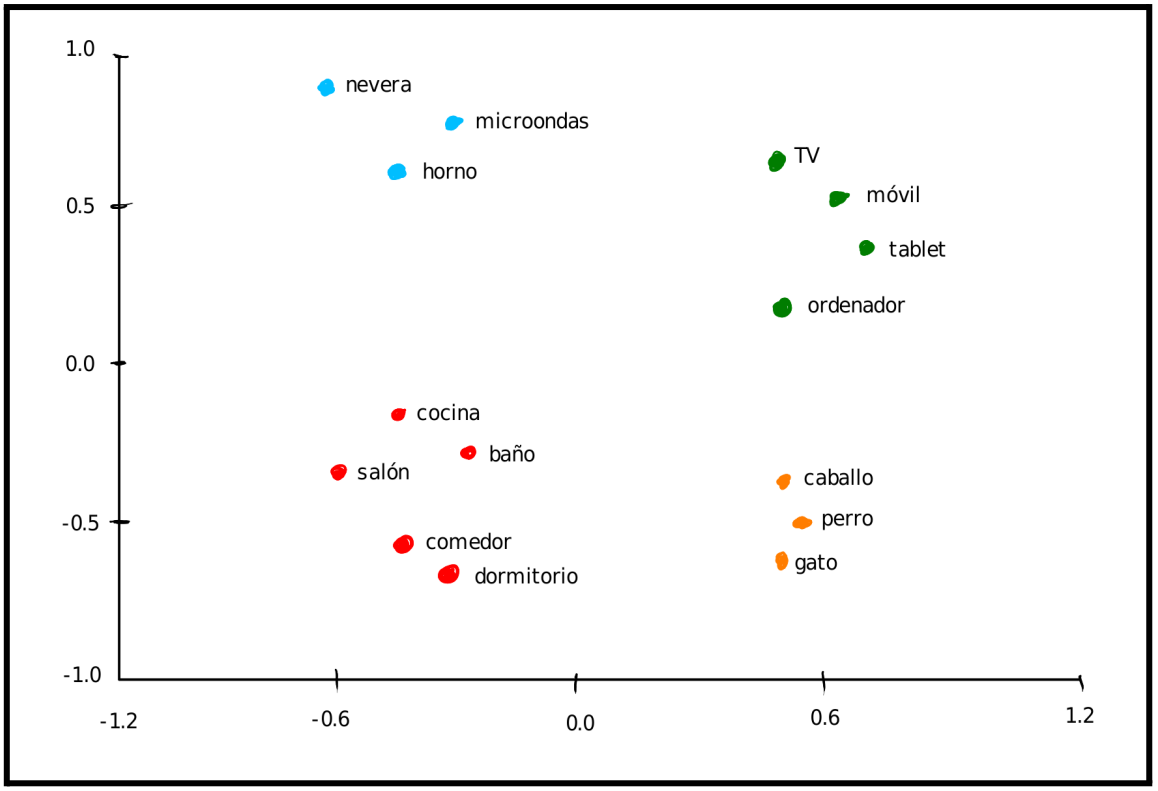


Figura 2.2: Ejemplo de *Word embeddings*

2.2 Detección de objetos

2.2.1 Introducción

En el ámbito del *machine learning* y de *computer vision*, la detección de objetos es una tarea de gran importancia. Sus objetivos principales son localizar los objetos de interés, definiendo unos cuadros delimitadores que los rodea, y clasificarlos dependiendo de sus características visuales.

Se utiliza en sistemas de vigilancia y seguridad, conducción autónoma, reconocimiento facial, entre otros.

Se fundamentan en modelos de aprendizaje profundo, en redes neuronales que procesan las imágenes y extraen las características necesarias para poder aprender a reconocer los objetos que les interesa.

Dentro de los pliegos se pueden encontrar expresiones matemáticas sencillas, de una línea, como puede ser una suma o una multiplicación, pero también expresiones complejas como fracciones o potencias. Dado que tienen un formato diferente, las librerías disponibles para procesar los PDF no son capaces de extraerlas. Haciendo pruebas se ha concluído que es mejor tratarlas como objetos dentro de imágenes que tratarlas como texto dentro de un archivo PDF. Por consiguiente, un modelo capaz de detectar objetos, como puede ser *YOLO,* es necesario dentro de la aplicación.

2.2.2 Redes neuronales convolucionales (CNN)

En este caso, al trabajar con imágenes, se usarán las redes neuronales convolucionales (*CNN*) [14]. Estas presentan un enfoque más escalable al clasificar imágenes y reconocer objetos por tener métodos para capturar características locales como bordes, texturas, y formas.

Las CNN están compuestas por varios tipos de capas :

- Capa convolucional
- Capa de *pooling*
- Capa totalmente conectada

La capa convolucional representa la primera capa de una red convolucional. Esta puede ir seguida por otras capas convolucionales o de *pooling*, pero la última necesariamente es una capa totalmente conectada. Con cada capa, la red aumenta en complejidad, identificando áreas cada vez más grandes de la imagen. Las primeras se enfocan en características más sencillas, como tonalidades y bordes. A medida que los datos de la imagen avanzan a través de la red, la CNN empieza a detectar elementos o formas más grandes hasta identificar el objeto deseado.

A continuación se detalla el funcionamiento de cada capa:

Capa convolucional

Es el bloque principal de una CNN, y es donde se llevan a cabo la mayoría de los cálculos. Se requiere de algunos componentes, como datos de entrada, un filtro y un mapa de características. La entrada puede ser una imagen en color, que será transformada en una matriz de tres dimensiones: altura, anchura y profundidad, que corresponden a la composición RGB en una imagen. Asimismo, se dispone de un detector de características, conocido como kernel o filtro, que se mueve por los campos receptivos de la imagen para verificar si la característica se encuentra presente. A este proceso se le conoce como convolución.

El tamaño del filtro suele ser una matriz de 3x3, aunque puede variar. Este se aplica a un área de la imagen con el fin de calcular el producto escalar entre los píxeles de entrada y el filtro. Dicho producto escalar se introduce en una matriz de salida. A continuación, el filtro se va desplazando y repitiendo el proceso hasta que haya recorrido toda la imagen. El resultado final se denomina como mapa de características, mapa de activación o característica convolucionada.

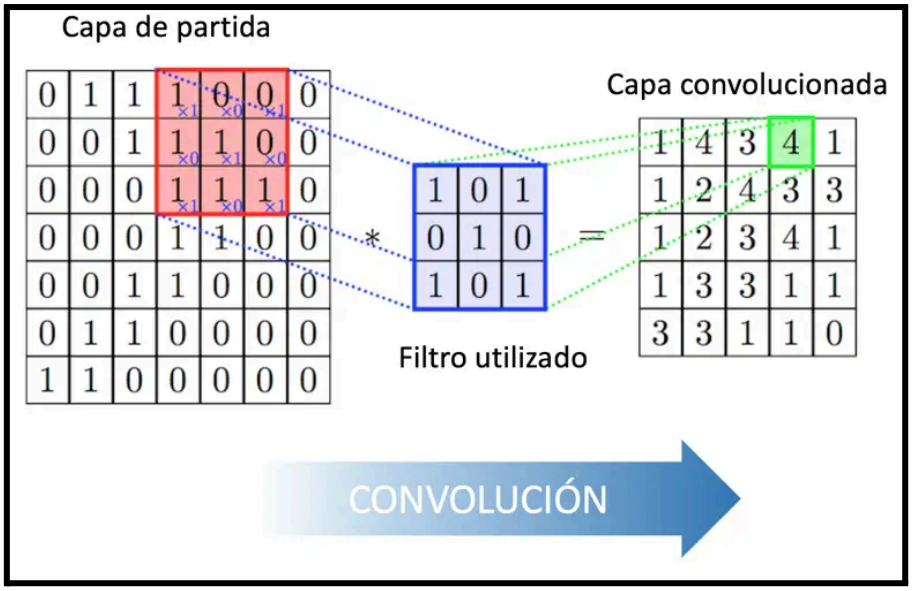


Figura 2.3: Ejemplo de un proceso de convolución

URL: https://www.diegocalvo.es/red-neuronal-convolucional/

Después de cada convolución, para introducir la no linealidad en la red, se aplica una capa de activación ReLU al mapa de características.

Como se ha comentado más arriba, la capa de convolución inicial puede ser seguida por otra capa de convolución. Al ocurrir esto, la estructura de la red puede volverse jerárquica, ya que las capas siguientes pueden ver los píxeles en los campos receptivos de las capas anteriores.

Capa de *pooling*

La agrupación de capas posibilita la disminución del número de parámetros de la entrada. De forma parecida a la capa convolucional, la operación de agrupación barre toda la entrada con un filtro, pero en este caso no tiene ningún peso. En su lugar, implementa una función de agregación de los valores dentro del campo receptivo y llena así la matriz de salida. Existen dos categorías principales de agrupación:

- Agrupación máxima (*Max Pooling*): conforme el filtro avanza por la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida. Ver figura 2.4. Este enfoque captura las características más destacadas (máximas activaciones) dentro de la ventana, lo que puede ser útil para preservar los bordes y texturas fuertes.
- Agrupación media (*Average Pooling*): a medida que el filtro recorre la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida. Este trata de suavizar la representación de características, manteniendo la información de manera más difusa en comparación con el *Max Pooling*.

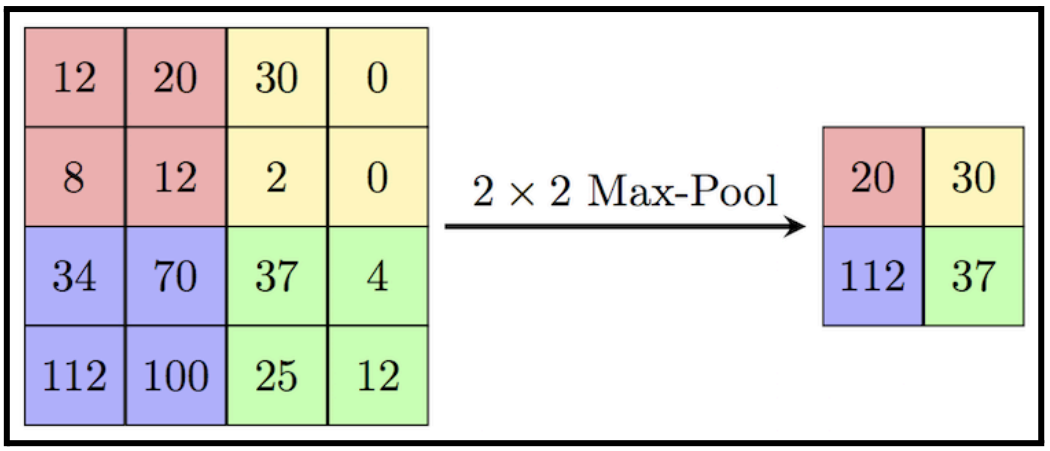


Figura 2.4: Ejemplo de un proceso de agrupación máxima

URL: https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/

Aunque se pierda mucha información en la capa, esta ayuda a reducir la complejidad, la cantidad de parámetros, y mantiene las características más importantes.

Capa totalmente conectada

Se sabe que los valores de píxel de la imagen de entrada no están relacionados de manera directa con la capa de salida. No obstante, en la capa totalmente conectada, cada nodo de la capa de salida se encuentra conectado directamente con un nodo de la capa anterior.

Esta capa efectúa la tarea de clasificación en función de las características extraídas de las capas anteriores. Utiliza una función de activación *softmax* para clasificar de forma adecuada los *inputs* y generar una probabilidad entre 0 y 1.

2.2.3 Entrenamiento del modelo

Mediante el entrenamiento del modelo de detección de objetos se consigue que este aprenda las características de los que deseamos que detecte. Posteriormente podrá reconocerlos en nuevas imágenes.

Al principio se tendrá que etiquetar manualmente un conjunto de imágenes de entrenamiento. Para cada una de estas se seguirán los siguientes pasos: definir mediante 4 coordenadas de la imagen un cuadro que delimite el objeto, y luego etiquetarlo con su clase respectiva.

A continuación se normalizará el tamaño de las imágenes y los valores de los píxeles. Con esto se conseguirá un entrenamiento más eficiente del modelo.

En cuanto a los parámetros de entrenamiento, se definirán los necesarios para cualquier red neuronal. Están las *epochs,* donde una *epoch* corresponde a una iteración completa sobre el conjunto de datos de entrenamiento. El *batch size* es el número de ejemplos de entrenamiento que se utilizan para actualizar los pesos de la red neuronal en cada iteración del algoritmo de optimización. Cuanto más pequeño es el valor del *batch,* más frecuentes

serán las actualizaciones. Por otro lado tenemos el *learning rate*, que determina el tamaño del paso que se da en la dirección del gradiente descendente durante la actualización de pesos del modelo. Para entender su utilidad, hay que introducir la función de coste, que es la medida de error del modelo. El objetivo del entrenamiento es minimizar dicha función, entonces si el valor del *learning rate* es muy alto, puede converger a un mínimo local subóptimo. En cambio, si el valor es muy bajo, el proceso de entrenamiento será más lento pero puede asegurar una convergencia eficiente a un mínimo global.

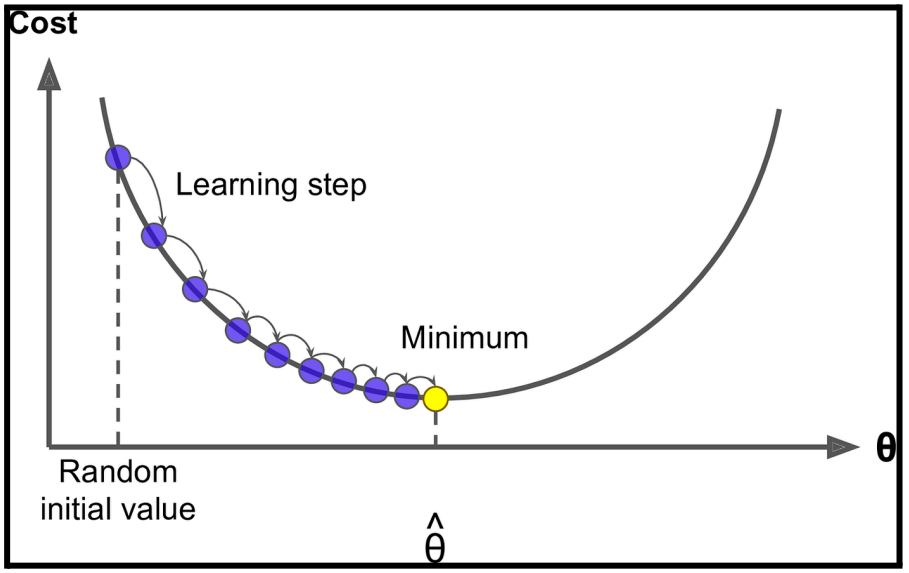


Figura 2.5: Proceso de convergencia a un mínimo global

URL: https://medium.com/@colemccracken/deep-neural-networks-choosing-a-learning-rate-172b97ef459

La red neuronal propiamente dicha se puede definir capa a capa, entrenándola desde cero, o se puede aplicar *transfer learning* sobre un modelo ya existente. Así se aprovecharía su aprendizaje previo y se podría aplicar a los nuevos objetos a detectar. De esta forma se reduce significativamente el tiempo y los recursos computacionales necesarios. Implica menos uso de hardware intensivo y por lo tanto una menor huella de carbono.

Como se ha explicado antes, la función de pérdida sirve para evaluar la precisión del modelo, calculando la diferencia entre el valor real y el valor que predice. Un optimizador es un algoritmo que reajusta los parámetros de la red utilizando el gradiente de la función de pérdida para poder minimizar su valor. Por ello hay que introducir también un optimizador.

Otro parámetro importante es el *Early Stopping*, que permite detener el entrenamiento de manera automática si los valores de la función de pérdida que se obtienen no disminuyen después de un número determinado de épocas. Puede significar el haber llegado a un mínimo global, aunque no se puede asegurar.

Una vez definidos los parámetros anteriores se puede ejecutar el entrenamiento para que la red neuronal comience a aprender las características de los objetos que interesan.

2.2.4 Medidas de rendimiento y corrección

Para evaluar el funcionamiento del modelo usaremos una serie de métricas que se basan en la Intersección sobre la unión (IOU). Esta métrica evalúa el porcentaje de superposición de dos cuadros delimitadores, uno que corresponde al cuadro que se ha definido manualmente al preparar las imágenes de entrenamiento y otro que corresponde al cuadro definido por el propio sistema. Cuanto más cerca de 1 es el valor de IOU, más fiable será el sistema de detección.

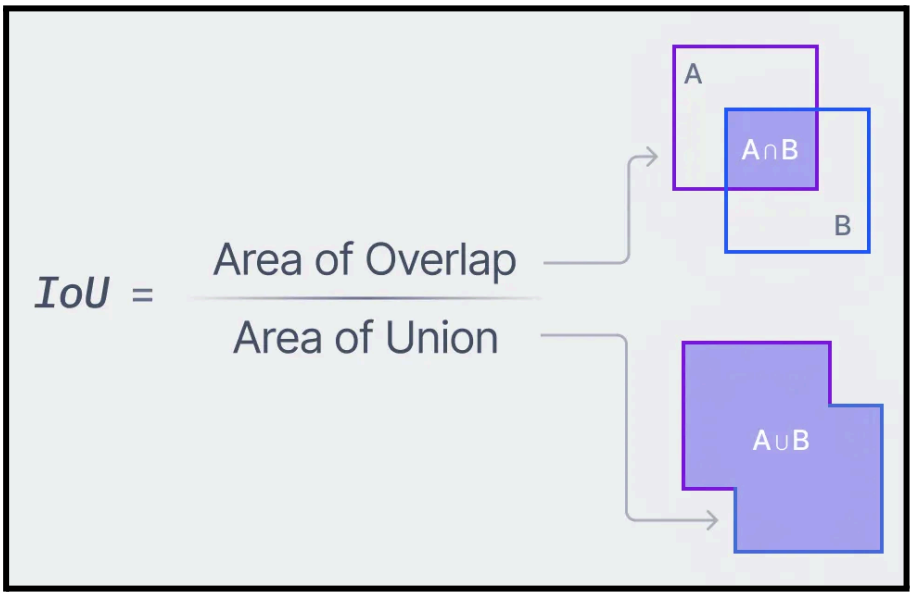


Figura 2.6: Representación gráfica del IoU

URL: https://www.v7labs.com/blog/intersection-over-union-guide

A partir del IOU se podrán definir los siguientes conceptos:

- **Matriz de Confusión**

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Figura 2.7: Ilustración de *Confusion Matrix*

URL: *https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/*

Para completar esta matriz, primero se tendrá que definir el valor de un *threshold*, que representa el mínimo valor de IOU para que se considere como una detección correcta. De normal son valores superiores a 50%.

Los campos de la matriz significan lo siguiente:

- ❖ Verdaderos Positivos (TP): Número de casos positivos correctamente identificados por el modelo.
- ❖ Falsos Positivos (FP): Número de casos negativos incorrectamente identificados como positivos por el modelo
- ❖ Verdaderos Negativos (TN): Número de casos negativos correctamente identificados por el modelo.
- ❖ Falsos Negativos (FN): Número de casos positivos incorrectamente identificados como negativos por el modelo.

Al tener un sistema de clasificación no binario, estos campos se traducirían de la siguiente forma. Para cada imagen del conjunto de test, se clasificará el resultado del sistema como True Positive (TP) si es una detección correcta y IOU > threshold. Se clasificará como False Positive (FP) si es una detección parcial y IOU < threshold. Será False Negative (FN) si no se detecta. True Negative (TN) no se utiliza en este caso ya que serían infinitos los cuadros delimitadores que no contienen objetos.

- **Curva ROC**
  Es una herramienta gráfica utilizada para ilustrar la relación entre la tasa de verdaderos positivos (True Positive Rate, TPR) y la tasa de falsos positivos (False Positive Rate, FPR).

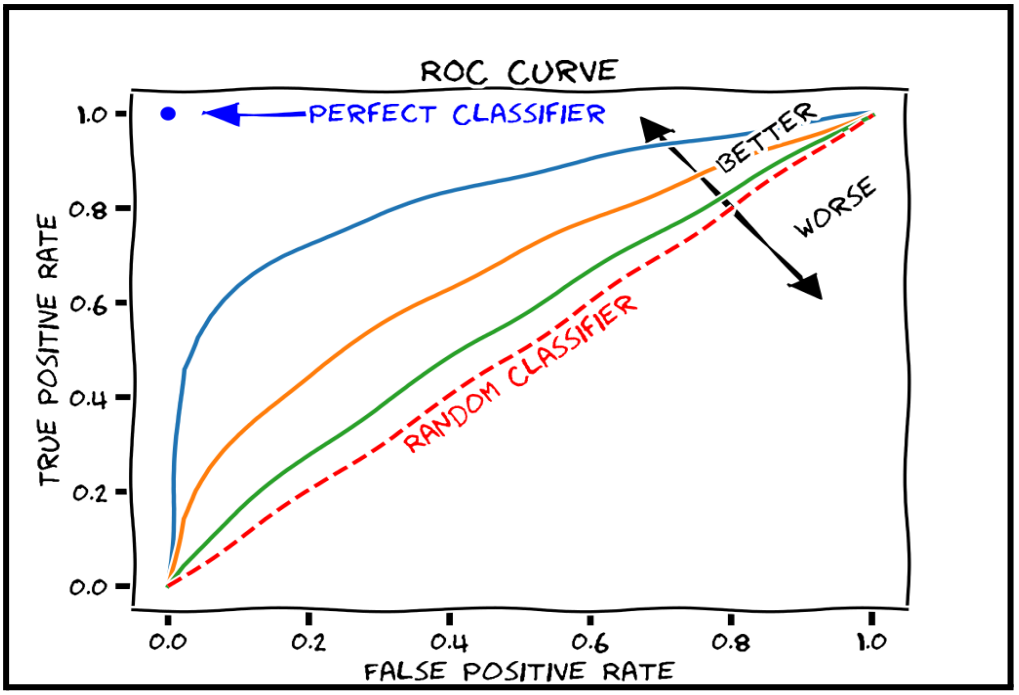


Figura 2.8: Ilustración de Curva ROC

URL: https://sefiks.com/2020/12/10/a-gentle-introduction-to-roc-curve-and-auc/

- **Precisión y *Recall***

  La **precisión** es la proporción de instancias clasificadas como positivas que realmente son positivas. En otras palabras, mide la exactitud de las predicciones positivas del modelo. Su valor ideal es 1.

  $$Precisión = \frac{TP}{TP + FP}$$

  Es útil en situaciones donde el costo de una clasificación incorrecta positiva es alto, por ejemplo, en diagnósticos médicos donde un falso positivo puede llevar a tratamientos innecesarios.

  El ***recall***, también conocido como sensibilidad o tasa de verdaderos positivos, es la proporción de instancias positivas que son correctamente identificadas por el modelo. Mide la capacidad del modelo para encontrar todos los casos positivos.

  $$Recall = \frac{TP}{TP + FN}$$

  Es útil en situaciones donde es crucial identificar la mayoría de los casos positivos, incluso si esto significa tener más falsos positivos. Por ejemplo, en la detección de enfermedades contagiosas, es más importante identificar todos los casos positivos para prevenir la propagación.

- **Curva de Precisión-*Recall***
  Los conceptos de precisión y *recall* están muy relacionados, ya que si se desea un sistema de mayor precisión, habrá una disminución de *recall*. Aplicado a la detección de objetos, se traduce en que el sistema detectará menos objetos ya que será más estricto. Por otro lado, para un sistema con más sensibilidad, se tendrá menos precisión, ya que deja de ser tan estricto. El concepto clave es el *threshold,* que ha sido mencionado anteriormente. Este decide el nivel de sensibilidad del sistema.

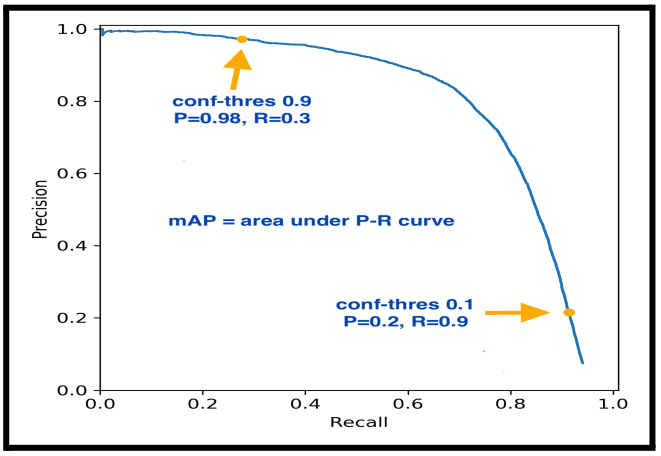


Figura 2.9: Ejemplo de Curva de Precisión-*Recall*

URL: https://github.com/ultralytics/yolov3/issues/898

  Un buen sistema será el que mantenga una precisión alta al aumentar el recall.

- **Precisión media (MAP)**
  Con esta medida se podrá valorar un sistema de detección de forma numérica, ya que esta corresponde al área bajo la curva de Precisión-Recall que ha sido explicada anteriormente.

  En la figura 2.9 se puede observar dicha área.

2.2.5 Ejemplo - YOLO

El algoritmo You Only Look Once (YOLO), es un sistema de código abierto para detección de objetos en tiempo real. Fue introducido en 2015 por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi en su trabajo de investigación "You Only Look Once: Detección de Objetos Unificada y en Tiempo Real". [17]

Este modelo hace uso de una única red neuronal convolucional para detectar objetos en imágenes. Como lo indica su nombre en inglés, requiere ‘ver’ la imagen una sola vez, lo que le permite ser muy rápido, pudiendo detectar fácilmente objetos en tiempo real en videos (hasta 30 FPS). Además, *YOLO* alcanza más del doble de precisión media (mAP) que otros sistemas, lo que lo convierte en un gran candidato para el procesamiento en tiempo real.

Para su funcionamiento, la red neuronal divide la imagen en regiones, prediciendo simultáneamente múltiples cuadros de identificación y probabilidades de clase de objeto por cada región.

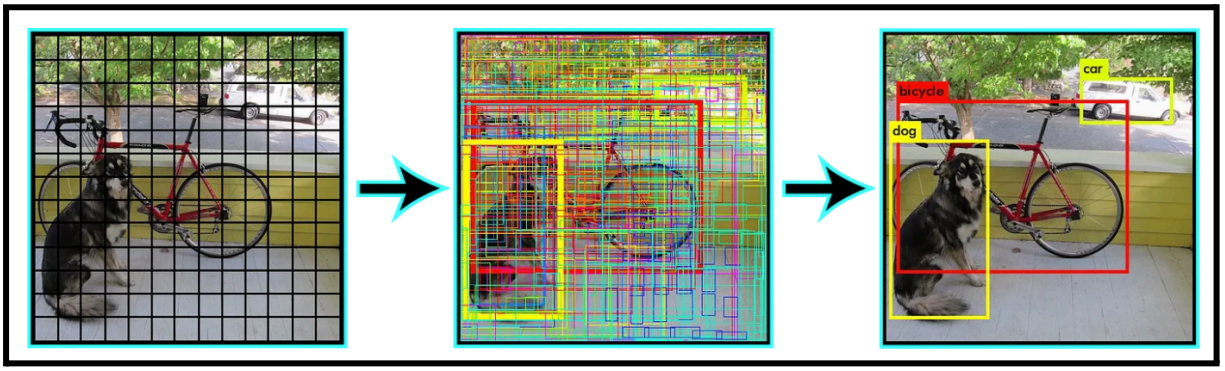


Figura 2.10: Ejemplo de detección de objetos del modelo YOLO

URL:
https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246

Como se puede observar en la figura anterior, el modelo primero divide la imagen en una cuadrícula de SxS (imagen de la izquierda). En cada una de las celdas predice N posibles *bounding boxes* y calcula el nivel de certidumbre (o probabilidad) de cada una de ellas (imagen del centro), es decir, se calculan SxSxN diferentes cajas, la gran mayoría de ellas con un nivel de certidumbre muy bajo. Después de obtener estas predicciones se procede a eliminar las cajas que estén por debajo de un límite. A las cajas restantes se les aplica un paso de *non-max suppression* que sirve para eliminar posibles objetos que fueron detectados por duplicado y así dejar únicamente el más exacto de ellos (imagen de la derecha).

Su arquitectura tiene en total 24 capas convolucionales, cuatro capas de agrupamiento máximo y dos capas totalmente conectadas. Su proceso es el siguiente:

- Se cambia el tamaño de la imagen de entrada a 448x448 antes de pasar por la red
- Se aplica una convolución 1x1 para reducir el número de canales, a la que sigue una convolución 3x3 para generar una salida cuboidal
- Estas capas, de 1x1 y 3x3, se van alternando hasta que en la última capa de convolución se genera un tensor 7x7x1024
- Tras la aplicación de las 2 capas totalmente conectadas, se obtiene un tensor de tamaño 7x7x30 como salida
- La función de activación es ReLU, excepto en la capa final, que utiliza una función de activación lineal
- Para calcular la pérdida YOLO usa el error de suma cuadrada (SSE) entre los cuadros delimitadores de las detecciones “bounding boxes” y los cuadros delimitadores de las anotaciones, que fueron definidos manualmente

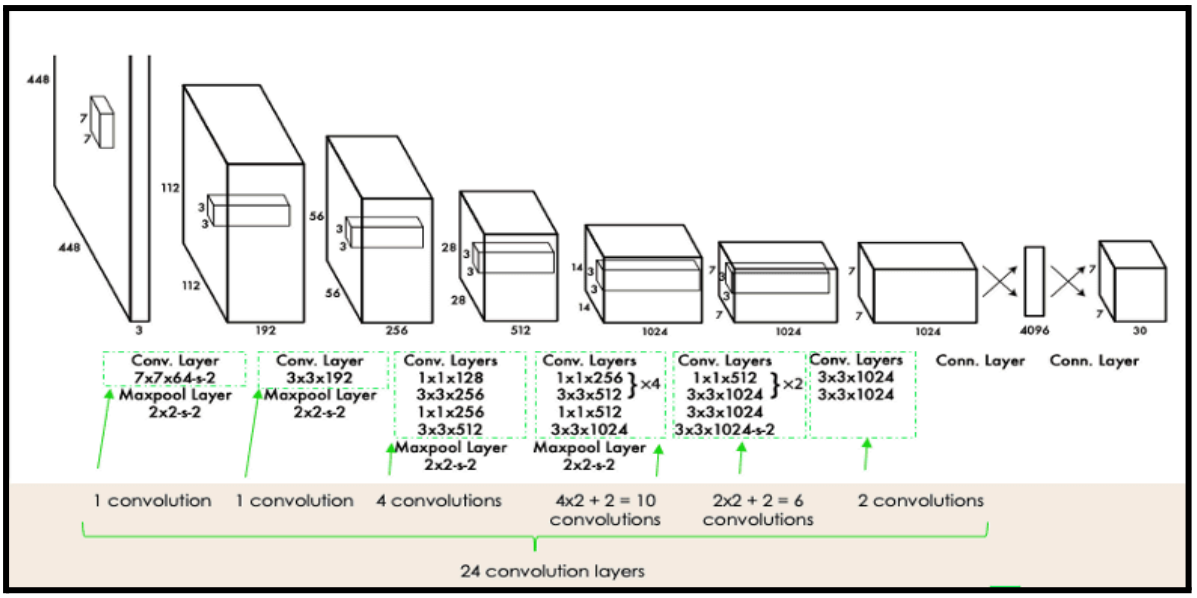


Figura 2.11: Arquitectura del modelo YOLO

URL: https://www.datacamp.com/es/blog/yolo-object-detection-explained

2.3 Redes neuronales comunes en Question Answering

2.3.1 Introducción

Las redes neuronales han permitido a los sistemas de QA avanzar desde modelos fundamentales de coincidencia de palabras clave hasta arquitecturas sofisticadas que permiten capturar la semántica y el contexto de las preguntas y respuestas. Esto ha sido posible gracias a varios tipos de redes neuronales, cada una aportando capacidades únicas para diversos aspectos del proceso que se sigue en un sistema de Question Answering.

2.3.2 Redes Neuronales Recurrentes (RNN)

Una red neuronal recurrente (RNN) [18] es un tipo de red neuronal que utiliza datos secuenciales o datos de series temporales. Se usan para tareas como la traducción automática, generación de texto, el reconocimiento de voz, entre otras. Se distinguen de otras redes neuronales por permitir que la información fluya de manera bidireccional, es decir, hacia atrás y hacia adelante a través de la red. Esto les permite mantener un estado interno y recordar información de secuencias anteriores.

Existen varios tipos de redes neuronales recurrentes, dependiendo del tamaño de la entrada y de la salida:

- ***One-to-many***
  La entrada es un dato único y la salida es una secuencia. Como ejemplo tenemos el *image captioning* donde la entrada es una imagen y la salida es una secuencia de palabras que la describen.

- ***Many-to-one***
  La entrada es una secuencia y la salida es una categoría. La tarea de clasificación de sentimientos tiene de entrada un texto y la salida será el sentimiento que se puede identificar.

- ***Many-to-many***
  La entrada y la salida serán ambas secuencias de palabras. Tenemos de ejemplo la traducción automática, donde dado un texto en un idioma se devuelve el mismo texto pero traducido.

En una *RNN*, cada capa oculta se determina a partir de la entrada actual y la salida de la capa anterior, como se muestra en la figura 2.12. Esto permite que la red capture información contextual de la secuencia. Además, en una *RNN* los parámetros que se comparten a lo largo de las diferentes etapas de la secuencia son las matrices de los pesos (de entrada W_x, recurrentes W_h y de salida W_y) y los vectores de sesgo (recurrentes b_h y de salida b_y).

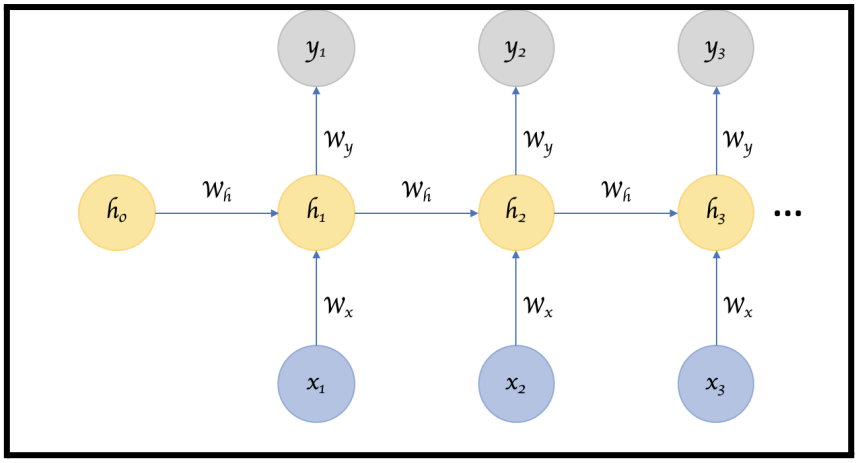


Figura 2.12: Esquema de las RNN

URL: https://juansensio.com/blog/034_rnn_intro

Se puede observar que los tres componentes fundamentales son los siguientes:

- **Capa de entrada X**
  Esta capa recibe los datos de entrada secuenciales. Si la secuencia es una oración, las palabras se representan como *embeddings*.

- **Capa recurrente H**
  En esta capa ocurre la retroalimentación, los estados ocultos actúan como memoria que se actualiza. En cada instante de tiempo *t*, la capa recurrente recibe el vector de entrada actual x_t y el estado oculto anterior h_{t-1}, y produce un nuevo estado oculto h_t.

  Matemáticamente se puede representar como:

  $$h_t = f(W_{xh} * x_t + W_{hh} * h_{t-1} + b_h)$$

  Donde:
  - W_{xh} es la matriz de pesos que conecta la entrada x_t con el estado oculto.
  - W_{hh} es la matriz de pesos que conecta el estado oculto anterior h_{t-1} con el nuevo estado oculto h_t.
  - b_h es el sesgo (bias)
  - f es una función de activación

- **Capa de salida Y**
  Esta capa toma el estado oculto h_t y lo transforma en una salida y_t correspondiente al instante actual.

  Matemáticamente se puede representar de la siguiente forma:

  $$y_t = g(W_{hy} * h_t + b_y)$$

  Donde:
  - W_{hy} es la matriz de pesos que conecta el estado oculto con la salida.
  - b_y es el sesgo (bias) de la capa de salida.
  - g es una función de activación

Las *RNNs* se basan en la propagación hacia adelante y hacia atrás del error de la red a lo largo del tiempo. Durante la propagación hacia adelante, la red procesa los datos de entrada en cada instante y calcula la salida adecuada. En cambio, durante la propagación hacia atrás, la red ajusta los pesos de la red para minimizar el error de la salida.

Una de las desventajas de las *RNNs* son los problemas de desvanecimiento y explosión del gradiente. Estos se producen cuando los gradientes usados para ajustar los pesos de la red son muy pequeños, en caso del desvanecimiento del gradiente, o muy grandes, en caso de la explosión del gradiente. Esto resulta en que los pesos de la red no se actualizan adecuadamente, dificultando la captura de dependencias a largo plazo. Para solucionarlo, se han propuesto varias variantes, como las *LSTM* y las *GRU*, cuya estructura es más compleja y mantiene mejor la información a largo plazo.

Memoria a corto-largo plazo (LSTM) [20]

Esta arquitectura es una variante de las redes neuronales recurrentes, que se enfocan en ampliar su memoria con el fin de aprender de experiencias importantes que han pasado mucho tiempo atrás.

Un ejemplo práctico es dentro de la predicción de la siguiente palabra. Tenemos la siguiente cadena: “En una tarde soleada de verano, los niños jugaban en el parque mientras los adultos se sentaban en los bancos, disfrutando de una conversación tranquila y observando a los pájaros que volaban sobre los árboles. De repente, uno de los niños vio algo brillante en el suelo y …". Al tener una frase muy larga, es más complicado que la *RNN* recuerde todas las palabras, y puede predecir palabras que no tengan nada que ver con el contexto anterior. Se le puede “olvidar” que es una tarde de verano, o que están en el parque.

Las *LSTM* permiten a las *RNN* recordar sus entradas durante un largo período de tiempo. Esto se debe a que contiene su información en una célula de memoria, sobre la que puede realizar distintas operaciones como leer, escribir y borrar información.

La arquitectura consta de varios componentes:

- La puerta de entrada (*input gate*) regula el flujo de nuevas entradas en la célula de memoria. Mediante una función de activación sigmoidea, se deciden los valores a conservar y los valores a descartar.
- La puerta del olvido (*forget gate*) utiliza también una función de activación sigmoidea para determinar la información que debe olvidar.
- La puerta de salida (*output gate*) es la que controla la salida de la célula de memoria. Además de la función de activación sigmoidea, se usa una función de tangente hiperbólica para determinar la información que se debe emitir.
- La célula de memoria es el componente principal, que almacena y olvida información de su estado interno.

En cada instante *t*, el modelo *LSTM* recibe un vector de entrada y un vector de estado oculto del instante anterior *t-1*. La *input gate* y la *forget gate* procesan el vector de entrada y el resultado se combina con el estado de la célula de memoria mediante una operación de suma de elementos. Por último, la *output gate* se utiliza para determinar la información que

debe salir de la célula de memoria y el vector de estado oculto resultante se pasa al siguiente instante temporal, *t+1*.

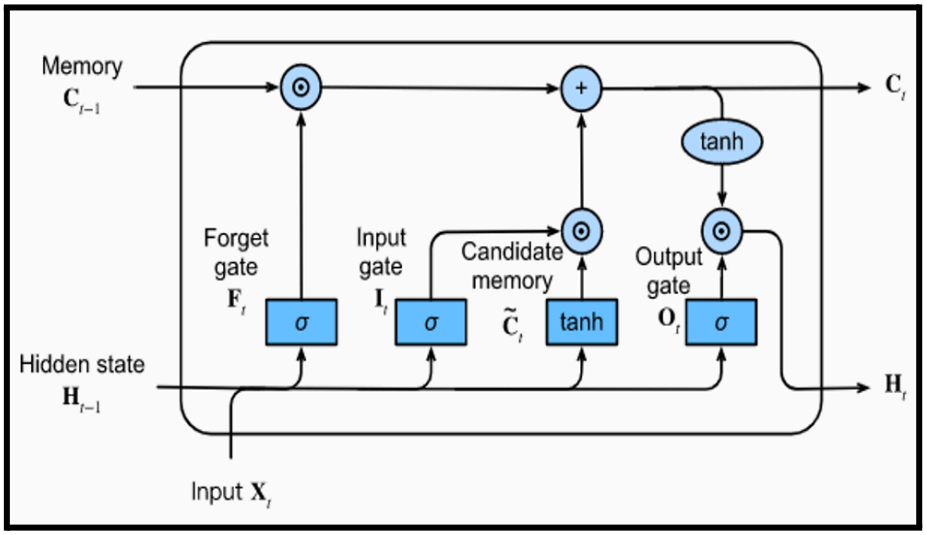


Figura 2.13: Arquitectura de la variante LSTM

URL: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

Gated Recurrent Units (GRU)

Las GRU combinan y simplifican algunos de los conceptos de las Long Short-Term Memory (*LSTM*), manteniendo la efectividad en el manejo de información secuencial. La principal característica de las *GRU* es la introducción de dos tipos de puertas: la puerta de actualización y la puerta de reinicio. Estas permiten que la *GRU* decida cuánto de la información anterior debe olvidarse y cuánto de la nueva información debe añadirse al estado oculto.

La compuerta de actualización determina cuánta información del paso de tiempo actual se debe almacenar en la memoria. Si la compuerta de actualización está cerca de 1, se mantiene una gran parte de la memoria anterior y se actualiza con la nueva información. Por lo contrario, si la compuerta de actualización está cerca de 0, se descarta la memoria anterior y se utiliza principalmente la nueva información.

De forma matemática se puede calcular así:

$$z_t = \sigma(W_z * [h_{t-1}, x_t] + b_z)$$

Donde:

- σ es la función sigmoide
- W_z es la matriz de pesos
- h_{t-1} es el estado oculto anterior
- x_t es la entrada actual
- b_z es el sesgo

La compuerta de reinicio decide cuánta información del paso de tiempo anterior se debe olvidar y cuánta se debe tener en cuenta en el cálculo de la nueva memoria. Si la compuerta de reinicio está cerca de 1, se mantiene gran parte de la memoria anterior. Si está cerca de 0, se descarta la memoria anterior y se inicia una nueva.

De forma matemática se puede calcular así:

$$r_t = \sigma(W_r * [h_{t-1}, x_t] + b_r)$$

Donde:

- σ es la función sigmoide
- W_r es la matriz de pesos
- h_{t-1} es el estado oculto anterior
- x_t es la entrada actual
- b_r es el sesgo

A diferencia de las *LSTM*, las *GRU* no tienen puerta de salida, por lo tanto son más simples y rápidas de entrenar. Al tener menos parámetros, también tienden a ser más eficientes y pueden generalizar mejor en algunos casos.

2.3.3 Transformers

La arquitectura de Transformers se basa en un modelo de aprendizaje profundo que aplica un conjunto de técnicas matemáticas para detectar las formas en que los elementos de datos en una serie influyen y dependen entre sí.

Se presentaron por primera vez en un documento de 2017 de Google [23]. Estos son una de las clases más nuevas y potentes de modelos inventados hasta la fecha, superando a las redes neuronales recurrentes (*RNN*) y las Long Short-Term Memory (*LSTM*) descritas anteriormente.

Se basan en el mecanismo de atención, permitiendo que el modelo preste atención a diferentes partes de la secuencia de entrada simultáneamente y determinar cuales son las más importantes. Esta capacidad de atención paralela y la eliminación de recurrencias hace que sea más eficiente, lo que le permite entrenarse en conjuntos de datos más grandes. También es más eficaz, especialmente cuando se trata de textos largos en los que el contexto lejano puede influir en el significado de lo que viene después.

A continuación se muestra una imagen de los componentes que forman parte de esta arquitectura:

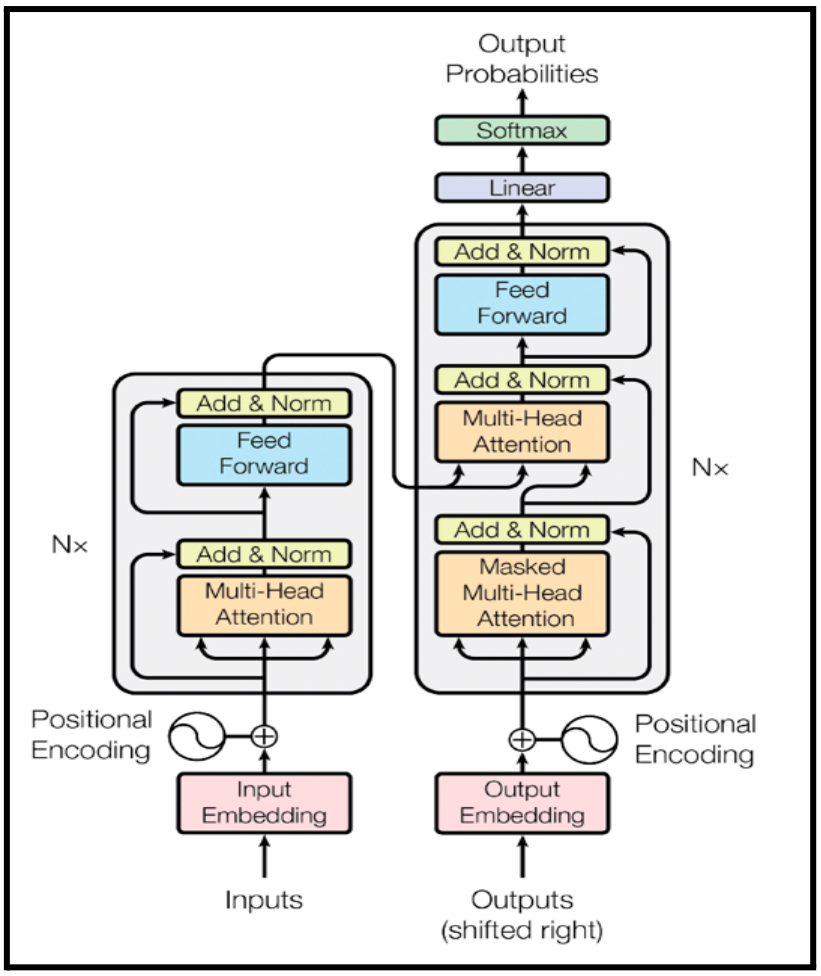


Figura 2.14: Arquitectura de los *Transformers*

URL: https://arxiv.org/pdf/1706.03762

El proceso que sigue es el siguiente:

- **Input Embedding:**
  Esta etapa consta de un tokenizador codificador de pares de bytes que segmenta la entrada de texto o imágenes en tokens. Cada uno se transforma en una secuencia vectorial matemática, mediante un word embedding.

- **Positional Encoding:**
  Los datos no son procesados en orden. Por ello se requiere una técnica para agregar información a la incrustación de cada token para indicar su posición en la secuencia. Esto se consigue mediante el uso de funciones que generan una señal posicional única. Así se puede comprender el contexto de la secuencia ya que se preserva el orden de los símbolos.

- **Transformer block:**
  Un bloque Transformer se compone principalmente de dos subcapas: un mecanismo de atención multi-cabeza y una capa *feedforward* completamente conectada.

La función de la atención autocentrada (*self-attention*) es determinar qué información de la entrada es relevante. Con el mecanismo de atención multi-cabeza (*Multi-Head Attention*), se calcula la atención varias veces en paralelo. Posteriormente, las salidas se concatenan y se proyectan de nuevo para obtener una salida, a la que se le aplica una capa de normalización.

Después se usa para que se aplique una red neuronal *feedforward* de dos capas a cada posición de la secuencia de manera independiente y en paralelo. También es seguida por una capa de normalización.

La normalización de capas mantiene las salidas de las capas dentro de un rango determinado que permite un entrenamiento más estable y evita problemas como el desbordamiento numérico y el gradiente explosivo o desvaneciente.

Dentro de los modelos *Transformer* es importante distinguir entre:

- *Encoder*: se compone de múltiples bloques *Transformer* en secuencia, donde cada bloque procesa la salida del bloque anterior.
- *Decoder:* Similar al *encoder*, pero con mecanismos adicionales de atención para manejar la entrada del *encoder* y la generación de la secuencia de salida.

- **Linear and softmax block:**
  El bloque lineal es otra capa totalmente conectada, también conocida como capa densa, antes de la etapa final. Realiza un trazado lineal aprendido desde el espacio vectorial hasta el dominio de entrada original. El modelo toma las representaciones internas y las convierte de nuevo en predicciones específicas que puede interpretar y utilizar. El resultado de esta capa es un conjunto de puntuaciones (*logits*) de cada *token* posible.

  La función *softmax* es la etapa final que toma esos *logits* y los normaliza en una distribución de probabilidad. Cada elemento de la salida de *softmax* representa la confianza del modelo en una clase o *token* en particular, por lo que si necesita hacer una predicción, como elegir la siguiente palabra de una secuencia, elegirá la palabra con mayor probabilidad.

Transformers - BERT

BERT(*Bidirectional Encoder Representations from Transformers)* [5]*,* publicado por Jacob Devlin y sus compañeros de Google en 2018, se basa en la tecnología Transformer. Destaca por poder ejecutar varias tareas propias del NLP sin ser entrenado para ninguna en específico, es decir, se entrena para que pueda interpretar el lenguaje en general y luego se puede especializar en una tarea en particular.

Esto se debe a su arquitectura interna que contiene dos grandes bloques. El primero es un bloque pre-entrenado basado en redes *Transformer* que permite el entendimiento general del lenguaje. El segundo bloque, concatenado al anterior, permite afinar el funcionamiento del sistema mediante técnicas de Deep Learning. Este segundo bloque es el que permite al sistema especializarse en una tarea en concreto.

A continuación explican detalladamente los dos bloques mencionados anteriormente:

- **Bloque de Pre-Entrenamiento:**
  BERT es pre-entrenado usando dos grandes conjuntos de datos en lengua inglesa: Wikipedia y google books, que en conjunto contienen más de 3.000 millones de palabras.

  Con este entrenamiento BERT aprende a analizar el texto de manera bidireccional, codificando cada palabra teniendo en cuenta su contexto tanto por la izquierda como por la derecha. Por ejemplo, una misma palabra puede tener dos significados distintos dependiendo de su contexto, y con esta bidireccionalidad BERT es capaz de codificar de manera precisa su significado.

  Para ello se lleva a cabo una tarea de enmascaramiento de lenguaje (*Masked Language Modeling, MLM)* en la que se ocultan aleatoriamente algunas palabras de la entrada y el modelo debe predecirlas en función del contexto, de las palabras que la rodean por la izquierda y por la derecha.

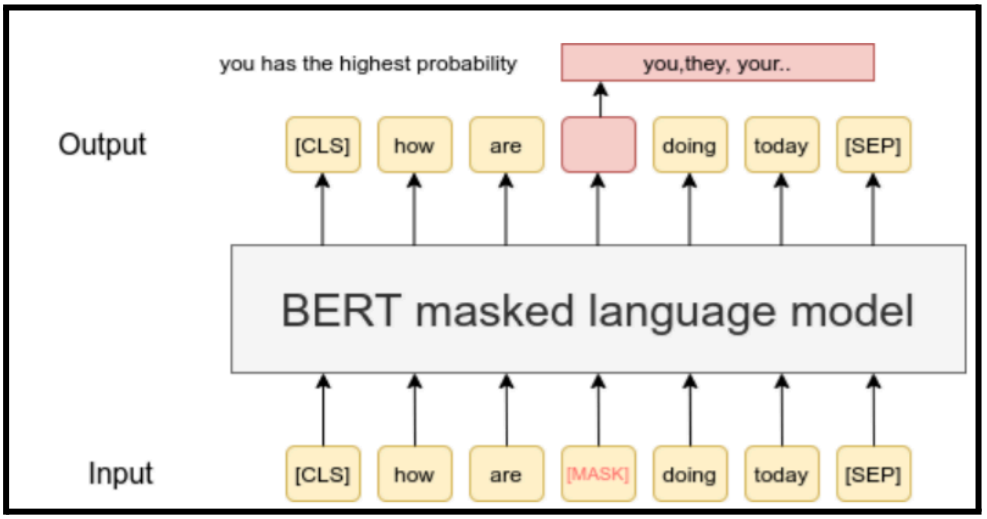


Figura 2.15: Proceso de enmascaramiento de lenguaje *MLM*

URL: https://www.sbert.net/examples/unsupervised_learning/MLM/README.html

Además de la tarea de MLM, también se le entrena con una tarea de predicción de la siguiente oración (*Next Sentence Prediction, NSP)*, donde el modelo debe predecir si una oración B sigue a una oración A en el corpus de entrenamiento. Para que BERT aprenda a identificar estas diferencias, durante el pre-entrenamiento el 50% de las veces la frase B es efectivamente la que sigue a A y el 50% de las veces restantes no lo es.

- **Afinación o *Fine-Tuning*:**
  Al modelo pre-entrenado se le agregan dos capas: una red neuronal y una capa *softmax*. Luego se presentan las frases de la tarea específica y se re-entrena el modelo de extremo a extremo. Este re-entrenamiento durará menos tiempo ya que el modelo total tiene un bloque ya pre-entrenado.

Cuando se tienen tareas como inferencia, equivalencia semántica, análisis de sentimientos o aceptabilidad lingüística, se tiene simplemente un problema de clasificación. Por ello a la salida de BERT se analizará únicamente la representación correspondiente al token de clasificación, y al ser bidireccional, se condensa toda la información necesaria para la clasificación en este único *token*.

Para las tareas tipo pregunta-respuesta se usa un esquema similar. La única diferencia es que al final la red neuronal y la capa *softmax* no entregarán una categoría, como en el caso anterior, sino que realizarán la predicción de los tokens de inicio y finalización de la respuesta encontrada dentro del texto.

BERT ha logrado un hito en el ámbito del procesamiento del lenguaje natural, brindando a las máquinas una comprensión más detallada y contextual del lenguaje. En 2019 se reveló que este modelo había sido incorporado a Google Search, permitiendo consultas en más de 70 idiomas.

Transformers - GPT

Los GPT (*Generative Pre-trained Transformer*) son una familia de modelos grandes de lenguaje basados también en la arquitectura de *Transformers*. El primero fue presentado en 2018 por OpenAI, una empresa de investigación y despliegue de inteligencia artificial que declara tener como misión asegurar que la inteligencia artificial general beneficie a toda la humanidad.

Los modelos GPT son redes neuronales pre-entrenadas en grandes volúmenes de datos de texto sin etiquetar, que son capaces de generar texto, contenido (imágenes, música, código y más), y entablar conversaciones de manera similar a como lo haría un humano.

La arquitectura de *transformers* estuvo disponible en 2017, lo que provocó la aparición de modelos grandes de lenguaje como BERT o XLNet. Estos eran transformadores pre-entrenados, pero no generativos, ya que eran solo codificadores (*encoders*).

El planteamiento semisupervisado que OpenAI empleó para crear un sistema generativo a gran escala consta de dos fases: una fase de pre-entrenamiento generativo no supervisado que establece los parámetros iniciales, y una fase de *fine-tuning* discriminativa supervisada para adaptar estos parámetros a una tarea en concreto. [16]

Durante el período de pre-entrenamiento, GPT experimenta una fase en la que se alimenta con una gran cantidad de texto de diversas fuentes, tales como páginas web, libros y artículos. Al procesar esta información, GPT adquiere patrones lingüísticos, gramática, semántica y una comprensión básica del mundo. De este modo se ajusta para predecir las palabras subsiguientes en una secuencia. A diferencia de BERT, que utiliza una pila de codificadores y se entrena bidireccionalmente, GPT se entrena de manera unidireccional, procesando el texto de izquierda a derecha.

Tras el pre-entrenamiento, llega el *fine-tuning*, en el que el modelo se adapta a tareas específicas como respuesta a preguntas, generación de texto, resumen de texto, etc. Este ajuste se realiza entrenando el modelo en datos etiquetados para la tarea específica.

Los chatbots son un modelo GPT muy relevante en la actualidad, ya que pueden llevar conversaciones como si de un humano se tratara. En el año 2022, OpenAI presentó el famoso ChatGPT, una interfaz de chat en línea impulsada por un modelo de lenguaje entrenado mediante RLHF, un aprendizaje por refuerzo a partir de la retroalimentación humana. Se trata de especialistas IA que proporcionaban conversaciones en las que interpretaban tanto al humano como al bot, y usaron estos diálogos para obtener un formato conversacional apropiado para un chatbot.

Los sistemas que se basan en *transformers* generativos se pueden orientar también a tareas que impliquen distintas modalidades. "Visual ChatGPT" de Microsoft combina ChatGPT con modelos de base visual (VFM), permitiendo imágenes como *input* y *output,* además de texto. Los progresos en la tecnología de conversión de texto a voz también ofrecen herramientas para crear contenido audio cuando se utilizan junto con modelos GPT.

Capítulo 3 - Desarrollo de la solución

Este apartado trata de explicar cómo se aplica la teoría explicada hasta ahora, para conseguir una herramienta web que cumpla con los objetivos planteados inicialmente. Cabe mencionar que hay distintas formas de desarrollar un sistema de *Question Answering*, pero dependen mucho de los tipos de textos que va a tratar y del nivel de exigencia que tienen las respuestas.

Se recuerda que el objetivo principal de este sistema es generar un informe a partir del contenido de los archivos correspondientes a una licitación. Estos contienen información tanto en formato de texto simple, como en formato de tablas y fórmulas matemáticas.

No existe una solución trivial para estas características. Las librerías que permiten extraer el texto de los archivos PDF son bastante limitadas en cuanto a los formatos permitidos. Por esa razón se ha pensado en soluciones *ad-hoc*, que permitan extraer el contenido de las tablas (mediante técnicas de OCR) y de las fórmulas matemáticas complejas (mediante un modelo de detección de objetos). Se han probado distintas tecnologías, intentando mejorar la corrección y la estética de las respuestas, pero se ha dado más peso a lo primero, ya que es más importante tener un sistema fiable. Más adelante se presentarán los pasos seguidos para extraer la información de cada uno de estos formatos.

También se va a explicar cómo se integra un modelo grande de lenguaje, *ChatGPT 3.5* o *Mistral Large*, para que pueda llevar a cabo la generación de respuestas para las preguntas que haga el usuario. Esas respuestas se basan en los documentos que proporciona un sistema de recuperación de información (*retriever*), que mediante una búsqueda semántica por *embeddings,* identifica los documentos más relevantes para una consulta, y los retorna al *LLM* para que los pueda procesar. La herramienta que conecta los elementos anteriores se llama *chain*, una clase de la librería 'langchain', que facilita la integración de modelos de lenguaje y de *retrievers* en aplicaciones de este estilo.

El informe final estará compuesto principalmente por unas preguntas ya predefinidas, que aportan información sobre los apartados más importantes de una licitación pública. La manera en la que están formuladas influye mucho en las respuestas que devuelve el *LLM*, por lo que se han refinado en múltiples ocasiones para conseguir los resultados esperados. También se mostrará la plantilla (*template*) que se ha usado para marcar los pasos que debe seguir el modelo de lenguaje para responder, consiguiendo así unos *outputs* más consistentes.

El usuario tendrá acceso a este sistema mediante una página web. Su interfaz es sencilla, ya que las únicas acciones que se podrán llevar a cabo serán la subida de archivos y la introducción de nuevas preguntas. También dispondrá de un método de autentificación para poder guardar un historial de informes de licitaciones, con la posibilidad de descargarlos en su sistema local en formato PDF. Se empezará por el desarrollo de este apartado ya que proporciona información visual al lector, lo que le ayudará a entender el flujo de la aplicación.

3.1 Desarrollo de la página web

Se usará una herramienta que permite crear aplicaciones web de manera rápida, ya que no hay muchos requisitos en cuanto a la interfaz gráfica del usuario. Esta se llama Streamlit, que es un marco de trabajo de código abierto en Python que facilita la creación de aplicaciones web interactivas y personalizadas para el análisis de datos y el aprendizaje automático.

Componentes

Dispone de múltiples widgets que permiten al usuario interactuar de forma dinámica con la aplicación. Entre ellos nos encontramos con el cargador de archivos, que permite subir archivos desde el sistema local, para poder ser procesados por el sistema. Se necesita para que el usuario pueda subir los archivos PDF del anuncio de la licitación, de los pliegos de cláusulas administrativas particulares y de los anexos correspondientes. Estos serán procesados por el sistema para poder devolver un informe.

Figura 3.1: *File Uploader* de Streamlit

Al recibir dicho informe, el usuario puede tener dudas acerca de algo que no se menciona en este. Para ello se dispone de una caja de texto, que le permitirá escribir su nueva consulta.

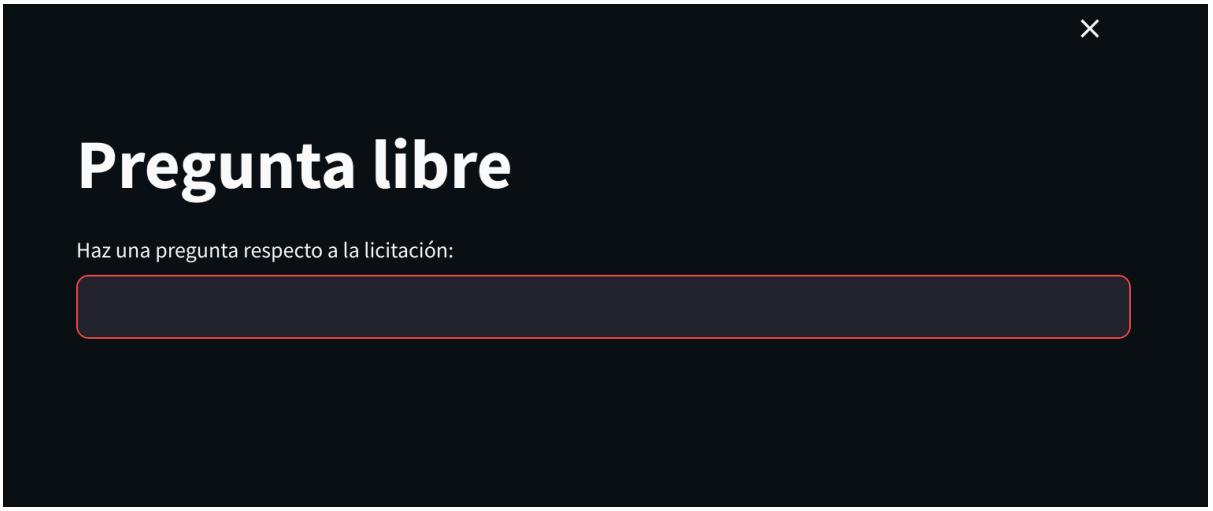


Figura 3.2: *Text Input* de Streamlit

Autentificación

Para que el usuario pueda mantener un historial de informes se dispone de un método de autentificación. Con la ayuda del componente 'streamlit-login-auth-ui', Streamlit se podrá conectar a una página pre-desarrollada que da el servicio de *Login/Sign-Up* de manera segura. Esta incluye también un método de restablecer la contraseña en caso de que al usuario se le olvida. Se le mandaría un correo con una contraseña generada aleatoriamente para que luego la pueda cambiar.

Esta librería también dispone de *cookies* encriptadas que permite al usuario acceder automáticamente a la aplicación sin la necesidad de iniciar sesión.

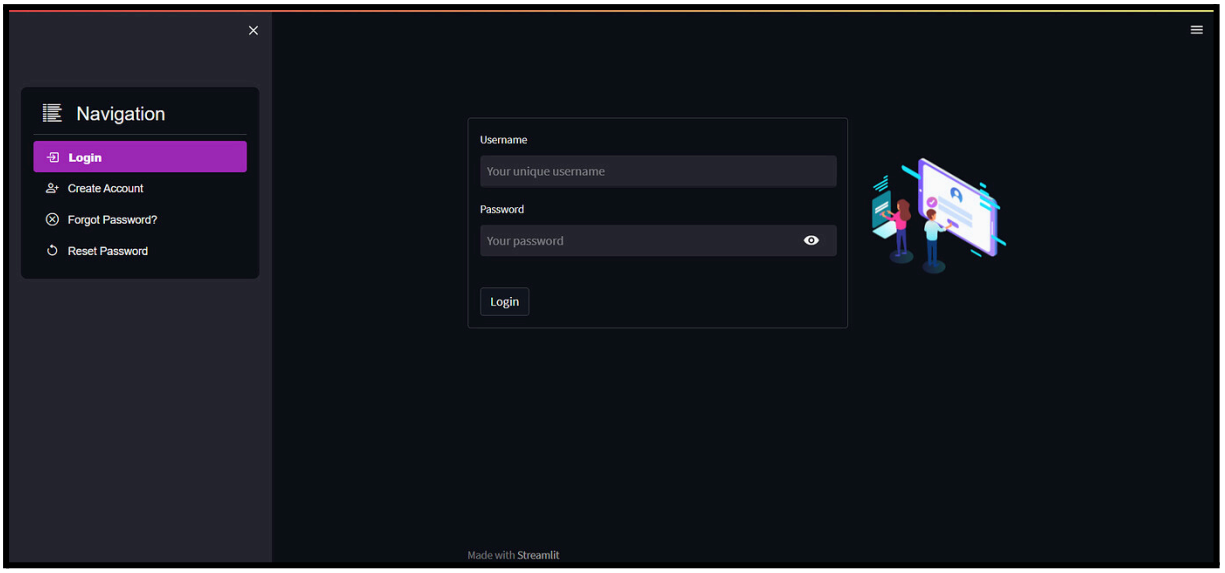


Figura 3.3: Proceso de autentificación

Después del *login*, aparecerá una barra lateral que incluye el historial de informes, teniendo de nombre al organismo contratante de la licitación. Al pulsar sobre uno de ellos, se mostrará en pantalla el informe completo y un botón de descarga, que le permite al usuario tenerlo en formato PDF en su sistema local.

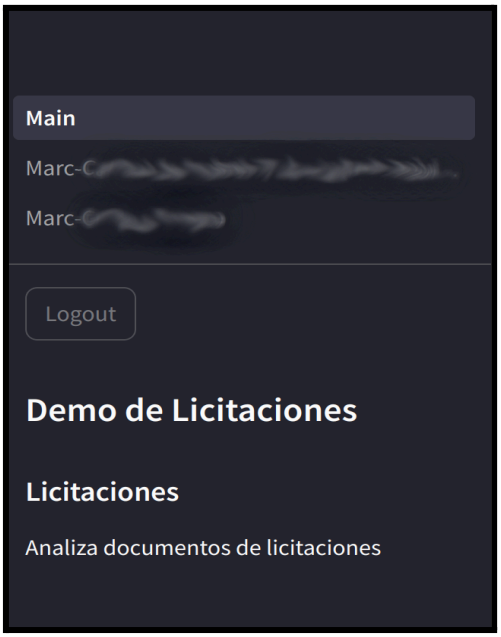


Figura 3.4: Histórico de informes

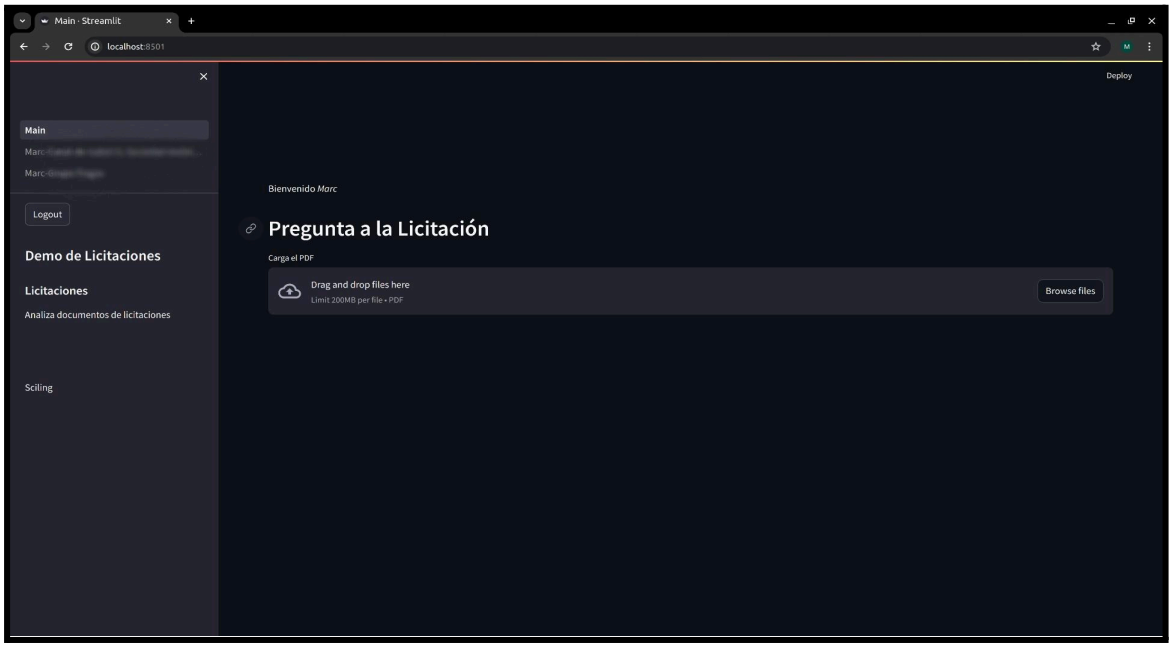


Figura 3.5: Interfaz completa del sistema

3.2 Extracción de texto simple

En este apartado se presentará el flujo que sigue el sistema desde la recepción de los archivos de los pliegos hasta la generación de las respuestas que serán incluidas en el informe final. Generalmente, todo sistema de *question-answering* sigue estos pasos, pero dependiendo de las necesidades de cada uno pueden cambiar ciertos parámetros o modelos de las tecnologías que se usan.

Como se ha mencionado anteriormente, el usuario dispone de un botón para subir todos los archivos correspondientes a la licitación. A continuación tendrán que ser procesados para convertir el contenido de estos a un formato óptimo para los modelos de recuperación de información.

Para ello se seguirán los siguientes pasos:

- **Extracción del texto de un archivo**
  Con la ayuda de la herramienta 'PdfReader', del paquete 'PyPDF2', se puede extraer todo el texto de los archivos. Se analizan iterativamente las páginas del PDF y se guarda el contenido en una variable de tipo *string*.

  Lo primero que se hace con este contenido es guardarlo con codificación UTF-8 en un archivo '.txt' en la base de datos. A priori se puede pensar que su identificador puede ser el mismo nombre que el archivo fuente PDF, pero tras analizar varias licitaciones se ha observado que se suele usar el mismo nombre para los archivos comunes. Por ejemplo, siempre hay un archivo 'anuncio.pdf', un archivo 'PCAP.pdf' y un archivo 'PPT.pdf'. Por lo tanto, si se usa únicamente el nombre como identificador, en la base de datos se reemplazarían constantemente.

  Por esta razón el nombre que se le pondrá será la combinación del nombre del archivo PDF de origen con un hash del contenido, consiguiendo un identificador único. Este se consigue con la ayuda de la librería 'hashlib', que usa la función hash criptográfica MD5 (*Message Digest Algorithm* 5) para producir un valor hash de 128 bits, y la función 'hexdigest' para obtener su representación hexadecimal.

  'PCAP_d5820616a554bd4ceae5222bb6728d09.txt' podría ser un ejemplo del nombre final de un archivo. Este no tendría conflictos con otros archivos llamados 'PCAP', al no ser que sean él mismo, ya que el hash ha sido calculado sobre el contenido que tiene dentro. Un archivo con el mismo nombre pero con un contenido distinto devolvería un identificador distinto, y se puede guardar sin ningún problema.

  A continuación, todo el contenido de tipo texto del archivo se tiene que dividir en bloques más pequeños llamados *chunks*. En un futuro interesa extraer solo los pasajes más importantes para la consulta del usuario, no todo el texto. También es importante mencionar que los modelos de lenguaje tienen un límite de ventana de contexto, por lo que es preferible descomponer el contenido del archivo en trozos más pequeños, para que puedan ser procesados. De eso se encargará un *text splitter.*

La división del contenido en bloques tiene su dificultad a la hora de decidir el tamaño del bloque. Por ejemplo, si es muy pequeño, puede separar un apartado importante del pliego en 2 bloques diferentes, lo que no es óptimo para luego generar una respuesta, ya que no estará la información completa. En cambio, si es muy grande, puede incluir demasiados temas diferentes y al *LLM* le puede costar más encontrar una respuesta.

Se tiene que buscar un tamaño de manera que un bloque tenga el contexto suficiente como para poder responder de manera precisa a la pregunta del usuario. Para este proyecto se ha optado por 5000 *tokens/chunk*, ya que es un valor que proporcionó un buen comportamiento. Además del tamaño (*chunk_size),* también disponemos de otro parámetro llamado *chunk_overlap*, que permite un solapamiento entre estos *chunks*, para que tengan un mínimo de consistencia entre ellos. A este se le ha atribuido el valor de 600 *tokens.*

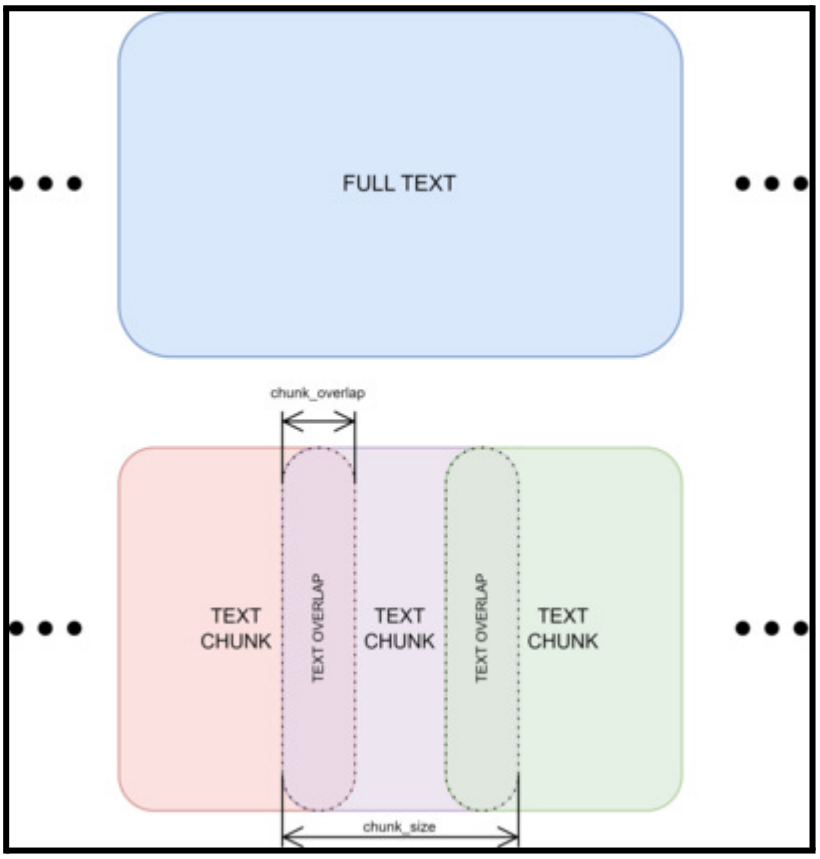


Figura 3.6: Representación de *chunk_size* y *chunk_overlap*

URL: https://youssefh.substack.com/p/hands-on-langchain-for-llm-applications-2d6

Existen varios *text splitters*, que pueden variar en la forma de realizar las divisiones, por carácter o por *token*. En este caso se ha usado la división por *token.*

Después de crear la lista de bloques, estos serán convertidos a la clase Document. Sirve para poder darles una mejor estructura, ya que también se les puede añadir metadatos, como el nombre del archivo, la página, sección etc.

Existe un método del sistema de recuperación de información, *retriever,* que puede usar estos metadatos para aplicarlos como filtros en su búsqueda. Por ejemplo, a los documentos que componen el archivo 'anuncio.pdf' se les puede añadir como metadato el nombre de este. Posteriormente, para alguna pregunta en concreto, se le puede especificar que busque solamente en ese archivo, y que no incluya información del 'PCAP.pdf' ni de otros anexos.

- **Generación de *embeddings***

  Como se explicó en el apartado del Estado del Arte, los embeddings son representaciones numéricas del texto (vectores), que sirven para encontrar textos que son semánticamente similares. Cuando el usuario haga una pregunta, se generará un vector que la represente numéricamente, y dentro del espacio de *embeddings* se buscará a los vectores más similares a este, que representarán los documentos que contienen la respuesta.

  Después de dividir el texto en bloques, que a su vez han sido transformados a documentos, cada uno será usado para crear su *embedding,* y luego serán almacenados en una Vector Store, ver figura 3.7. Esta es una base de datos diseñada para almacenar y gestionar representaciones vectoriales de datos. Posteriormente será usada para encontrar los documentos más relevantes para la pregunta del usuario.

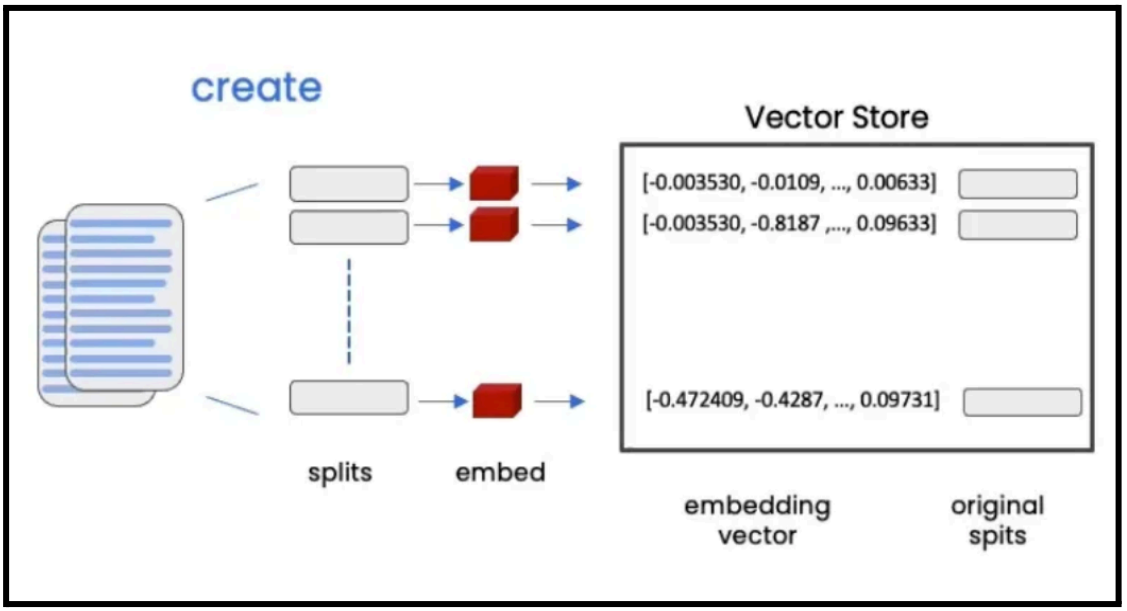


Figura 3.7: Almacenamiento de los *embeddings* en Vector Store

URL:
https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed

Para este proyecto se ha optado por el uso de modelos pre-entrenados de OpenAI para crear los *embeddings* a través de su API. Existen varios, con sus respectivos costes y rendimientos, que se probarán con el objetivo de encontrar el mejor en relación calidad/precio.

Para la Vector Store existen varias herramientas que se pueden usar. Entre las más comunes se encuentran FAISS y ChromaDB, pero en este proyecto se usará FAISS (Facebook AI Similarity Search). Es una biblioteca de código abierto desarrollada por Facebook AI Research (FAIR) para la búsqueda eficiente de similitud y el clustering en vectores de alta dimensión. Una de sus ventajas es que puede aprovechar la potencia de las GPUs para acelerar la búsqueda y el procesamiento.

Para evitar la creación de *embeddings* para archivos que ya han sido analizados, se puede usar un método de la Vector Store para almacenarlos en un directorio persistente. Se usará el formato 'Pickle', que sirve para serializar y deserializar estructuras de datos. Los *embeddings* de cada documento se convierten en una secuencia de bytes y se almacenan en un archivo '.pkl'. Cuando el sistema los necesita recuperar del disco, se utiliza un proceso inverso de deserialización para convertir esa secuencia de bytes a un objeto en memoria.

Cada archivo tendrá un identificador único, y cada vez que se suba un PDF nuevo a la aplicación se comprobará si sus *embeddings* ya se encuentran en disco, para evitar crearlos nuevamente. De esta manera no se hacen llamadas innecesarias a la API de OpenAI y por lo tanto no se gasta más dinero.

- **Extracción de *chunks* relevantes (*Retrieval*)**
  La corrección de las respuestas que dará el sistema depende en gran parte de esta fase. Un *retriever* es una interfaz encargada de encontrar los bloques de información (documentos de la Vector Store) que contienen la información necesaria para responder a una pregunta en lenguaje natural, que generalmente proviene del usuario. En la siguiente fase, estos documentos serán el *input* para el modelo de lenguaje, ya que este se tiene que basar en ellos para generar su respuesta.

  La Vector Store, que hasta ahora era un almacenamiento de *embeddings*, también puede actuar como *retriever*. No obstante, hay muchos más tipos, que pueden usar un *LLM,* que pueden ordenar los documentos retornados, o que combinan varios *retrievers* en uno.

  El que incluye un *LLM*, llamado '*Self Query'*, es capaz de reformular la consulta inicial y conseguir nuevas consultas distintas pero con la misma intención. De esta forma hace una petición con cada una de las nuevas *queries*, que devolverán unos documentos, y la intersección de estos será el resultado final del *retriever*.

  En cambio, el *retriever 'Long-Context Reorder'* se encarga de reordenar los documentos que devuelve, ya que normalmente se les da menos atención a los que se encuentran en el medio de la lista de documentos. Por lo que su objetivo es poner los más relevantes al principio o al final de la lista, para que posteriormente el *LLM* que los procese les dé más atención.

Existe también la posibilidad de usar varios *retrievers,* y combinar sus resultados. 'Ensemble' es capaz de hacerlo, poniendo distintos pesos a cada *retriever* que use, pudiendo conseguir una 'búsqueda híbrida'. Esta consiste en juntar una búsqueda semántica con una búsqueda basada en palabras clave. Se usa un *retriever* de cada tipo, poniendo más peso al que más conviene, y de esta forma se puede buscar por la intención detrás de la consulta y por las palabras clave de la misma.

A pesar de que la librería 'Langchain' ofrezca tantas posibilidades, se ha optado por usar la opción más común: la Vector Store. Esta ha devuelto buenos resultados sin realizar pasos extra como las demás, lo que ahorra en tiempos de ejecución y en llamadas a la API del *LLM*.

Existen varios métodos de la Vector Store para recuperar los documentos más relevantes para una consulta. Uno de ellos es por *Similarity Search*, que es el más básico, y se le pasa la *query* junto a un valor *k*, que representa el número de documentos relevantes que debe devolver. La *query* se convierte a un vector numérico y la Vector Store calcula la similitud entre el vector de consulta y cada uno de los vectores en el conjunto de *embeddings* utilizando una métrica de similitud específica, como la distancia coseno o la distancia euclidiana. Después, teniendo los *k* vectores más similares de la Vector Store, se devuelven los documentos asociados a cada uno de ellos. En teoría, ahí se debe encontrar la respuesta a la pregunta del usuario. Pero en la práctica pueden surgir inconvenientes, como la falta de diversidad.

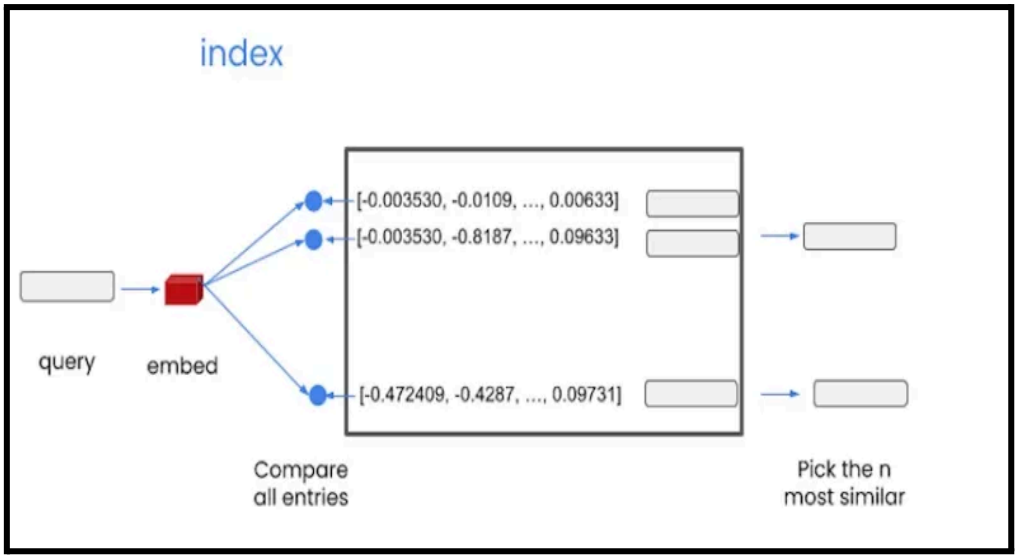


Figura 3.8: Recuperación de documentos mediante *Similarity Search*

URL:
https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed

Para ello está la posibilidad de usar otro método como *Maximum Marginal Relevance* (MMR), que fuerza esa diversidad en los resultados de la búsqueda. Es útil para poder devolver información que no solo sea semánticamente similar a la

consulta, sino que pueda servir para un contexto más general. Aparte de la *query*, se le pasará como argumento un valor de *fetch_k,* que representa el número de documentos que debe recuperar en un punto intermedio, que tenga tanto relevancia para la pregunta como diversidad. De ese conjunto, se elegirán los *k* más diversos para la respuesta.

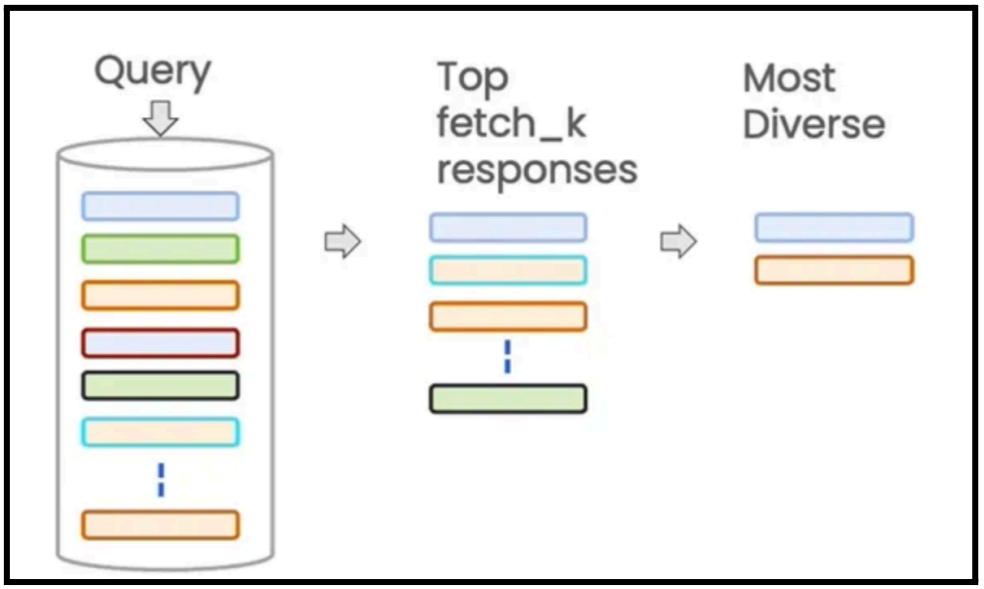


Figura 3.9: Recuperación de documentos mediante *MMR*

URL:
https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed

Además, también existe la posibilidad de filtrar por metadatos. Por ejemplo, al generar los distintos *chunks* y convertirlos a documentos, se les puede añadir los metadatos que se consideren convenientes. La Vector Store se puede usar para recuperar documentos relevantes para la consulta, pero esta vez filtrados por dichos metadatos, como pueden ser el nombre del archivo fuente, la página, sección etc.

También existe una forma de comprimir los documentos con el objetivo de eliminar información innecesaria. El método de 'Contextual Compression' se basa en recorrer todos los documentos con un modelo grande de lenguaje para extraer los pasajes más relevantes, para poder devolver solo estos y no los documentos completos.

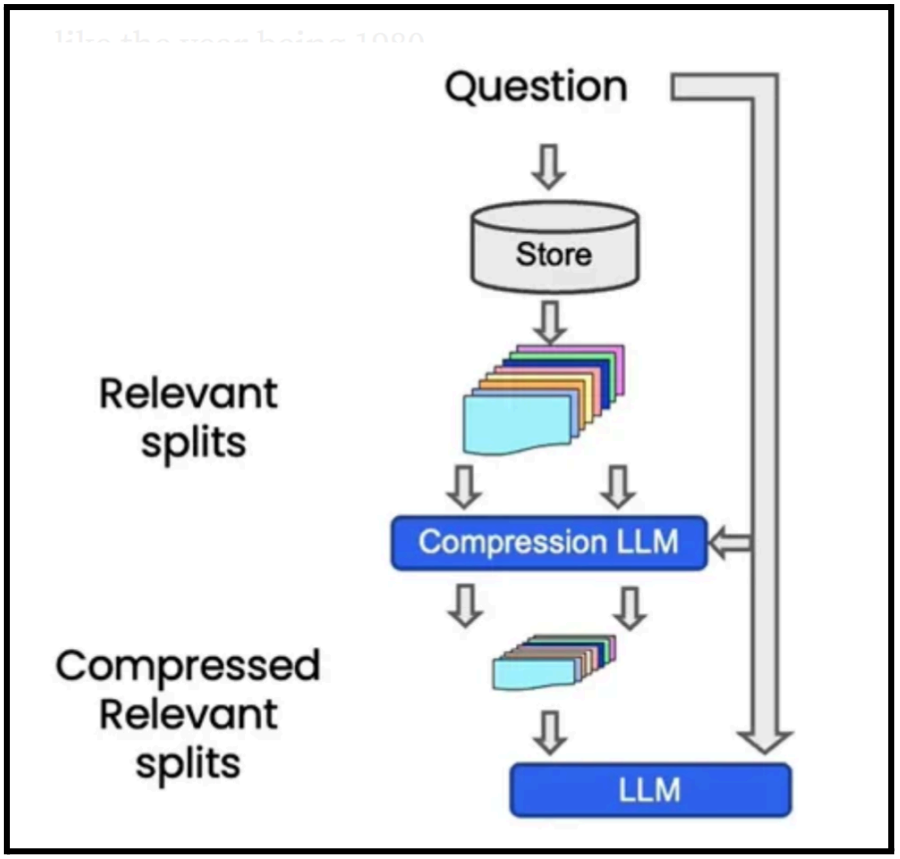


Figura 3.10: Recuperación de documentos mediante Contextual Compression

URL:
https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed

En el desarrollo de este proyecto se han probado todos los distintos enfoques anteriores pero se han conseguido los mejores resultados con el primero, con el *Similarity Search.* A pesar de ser el más básico, funciona bien para las características del sistema. Como parámetro *k,* número de documentos que retorna, se le ha pasado 10, pero este valor irá variando dependiendo de la ventana de contexto del LLM.

Cada documento contiene un número específico de *tokens*, y si el *retriever* recupera muchos documentos que contengan muchos *tokens*, es probable que se supere el límite que puede procesar el modelo de lenguaje. Para prevenir este problema, se ha desarrollado un método que calcula constantemente el número de *tokens* recuperados por el *retriever*, haciendo que este se mantenga por debajo del límite.

- **Generación de respuesta con el LLM**
  Teniendo la pregunta inicial del usuario y los documentos que contienen la información necesaria para responderla, devueltos por el *retriever,* el modelo de lenguaje debe ser capaz de generar una respuesta coherente.

  Se han probado diferentes modelos para ver cuál es el LLM más “inteligente”. Además de los que ofrece la API de OpenAI, también se han probado los de

MistralAI y de Cohere. Estos últimos ofrecen tanto API para hacer las llamadas y computaciones en sus servidores, como los pesos de los modelos en Hugging Face, para poder desplegarlos en local.

Para integrarlos a nuestra aplicación se usa 'LangChain', una biblioteca de código abierto diseñada para facilitar el desarrollo de aplicaciones que utilizan modelos de lenguaje, como los de procesamiento de lenguaje natural (NLP).

Con esta biblioteca se podrá crear una *chain*, que es la encargada de conectar la pregunta del usuario, los documentos relevantes y el *LLM*. A continuación se explicarán los distintos argumentos que se la pasan a la hora de crearla:

- **LLM:** Se define el modelo de lenguaje que se usará para procesar las preguntas y para generar las respuestas. Existe la posibilidad de definir algunos apartados de la configuración del modelo, como la temperatura o la *seed*. La temperatura es un hiperparámetro que controla la aleatoriedad de los resultados. Una temperatura alta produce resultados más impredecibles y creativos, mientras que una temperatura baja produce resultados más comunes y conservadores. Para este proyecto se ha decidido por un valor 0 de temperatura. Además, para que el modelo produzca los mismos resultados cada vez que se ejecute con un mismo input, también se establece una *random seed* fija. De esta forma se podrán depurar errores y optimizar el sistema, ya que aporta consistencia en las respuestas, y elimina lo máximo posible la aleatoriedad propia de los sistemas que implementan modelos de lenguaje.

  Dependiendo del *LLM* que se usa, se tendrá una ventana de contexto u otra. Esta es la suma de tokens que tienen: el input del usuario, los documentos devueltos por la Vector Store y la respuesta que genera el propio modelo. Los que se han usado para este proyecto son de 16k *tokens*, por lo que se ha tenido que controlar de alguna forma el contexto que se le pasa al modelo para no superar su límite.

  Para ello se hace uso de la librería 'tiktoken', que es un *tokeniser* de los modelos de OpenAI. Por cada documento recuperado por la Vector Store se usa esta herramienta para contar los *tokens* presentes. Cuando la suma de *tokens* exceden el límite de 16k, deja de recuperar documentos, por lo que se mantendrá siempre en el intervalo permitido. Por esto el valor *k* que se pasa al método de Similarity Search no es constante, sino que depende en cada momento de la suma total de *tokens*.

- **Tipo de *chain*:** Existen muchas variaciones de *chains*, dependiendo de cómo manejan los documentos que se le pasan de input y cómo va generando la respuesta final. En esta aplicación se usa la *chain* de tipo '*stuff',* que junta todos los documentos en uno y se la pasa al *LLM* para que lo procese, con una sola llamada a la API. Como se ha comentado antes, se tiene que controlar la ventana de contexto, por lo que los tamaños de todos esos documentos ya están previamente calculados para que no excedan el límite

de *tokens*. En cambio, si no estaba disponible ese pre-cálculo de contexto, también se podían usar cadenas de tipo “reduce”, “map reduce” o “refine”. Estas se encargan de dividir los documentos en *chunks* más pequeños para que puedan entrar en la ventana de contexto del *LLM*, pero significa un mayor gasto ya que se harían múltiples llamadas a la API para cada *chunk*.

- **Prompt Template:** Estas plantillas permiten a los desarrolladores definir los pasos que debe seguir el modelo para poder generar una respuesta. Se proporciona una estructura fija para que el input del *LLM* tenga siempre un formato determinado, y las únicas variables en este caso serán la pregunta del usuario y el contenido de los documentos relevantes a la pregunta.

  Para guiar de alguna forma el modelo del sistema, se ha definido el siguiente *template*:

```
'system_template' :
"""
Tarea: Necesito que extraigas y presentes información de documentos de forma clara y directa.

Estilo: Tu explicación debe ser clara y al grano, sin entrar en conversaciones o preguntar.
No hagas introducciones, menciona los datos directamente.

Cómo responder: Simplemente responde a lo que se pregunta sin repetir la pregunta.

Cuando no sepas algo: Si no tienes la respuesta o te falta información, di claramente que no
lo sabes. Evita dar referencias o inventar algo.

Sobre las referencias: No menciones de dónde sacas la información, no menciones el número de
página o el archivo PDF.

Idioma: Todo esto, por favor, en español.
----------------
{context}
----------------
"""
```

Figura 3.11: *Template* definido para el sistema

La variable final *context* se refiere al contenido de los documentos retornados por el *retriever*, y junto a la pregunta del usuario, se le pasará todo al *LLM*.

Se pueden observar las pautas que se le definen para llevar a cabo el proceso de generación de respuesta.

Un modelo grande de lenguaje es muy impredecible, y por eso se le tiene que indicar su tarea de la manera más concisa posible. Se le pide no responder con información que no se encuentre en los documentos o que sea innecesaria. Al hacer pruebas, se observó que este era capaz de inventar cosas, por falta de contexto, para poder responder plenamente a las preguntas del usuario.

A pesar de que este es capaz de identificar el idioma de los documentos, se le ha tenido que mencionar que responda siempre en castellano. En algunas ejecuciones se podía equivocar y responder en inglés, que es el idioma en el que está mayoritariamente entrenado.

Para este proyecto se han probado exclusivamente licitaciones en castellano, pero debería ser capaz de procesarlas también en otros idiomas, aunque no se asegura la misma capacidad y fluidez en sus respuestas.

Por último queda usar la función *run* de la *chain*. Pasándole como argumentos la lista de documentos relevantes para la consulta del usuario y la consulta en sí, generará una respuesta de acorde a los parámetros establecidos anteriormente, al crear la cadena.

3.3 Extracción de tablas

Existe la posibilidad de que el usuario introduzca archivos correspondientes a anexos de la licitación, y estos pueden contener información en formato de tabla. La librería de lectura de los PDF presentada antes, PyPDF2, no es capaz de extraer con precisión la estructura de la tabla, sino que extrae el texto en crudo, por lo que pierde información importante. Para ello usaremos unas herramientas extra, *Unstructured* y *Tesseract OCR*.

La biblioteca *unstructured* de Python proporciona métodos avanzados para el procesamiento y la extracción de datos de documentos, incluyendo tablas de archivos PDF. Por otra parte, *Tesseract OCR (Optical Character Recognition)* se basa en analizar el texto contenido en una imagen y lo convierte en texto editable utilizando técnicas de reconocimiento de patrones y algoritmos de aprendizaje automático. La imagen se divide en zonas que contienen texto. Esto puede incluir la identificación de párrafos, tablas, líneas y palabras individuales.

Estas dos herramientas trabajan en conjunto para que en el método de 'partition_pdf' de la librería *unstructured*, se le pueda definir la estrategia conveniente, en nuestro caso "auto". Con ello, si la librería identifica solo texto en las páginas del anexo, no usará un proceso de OCR para extraer tablas porque no las hay. Así el procesamiento será más rápido. En cambio, si existen tablas y el parámetro de 'infer_table_structure' está con valor 'True', sí que aplicará OCR y guardará un documento con el contenido en texto por un parte y la tabla con sus componentes por otra parte. Su estructura se guarda en formato html, por lo que el formato de la tabla será una variable de tipo *string,* con los *tags* correspondientes como '<table>', '<tr>', '<td>' etc.

El inconveniente de este enfoque es el idioma, ya que mayoritariamente el OCR está entrenado para el inglés, y en palabras con acentos o con 'ñ' puede retornar valores incorrectos.

El momento en el que se aplica esta función de extracción de tablas es cuando la respuesta del LLM a alguna consulta de un usuario incluye el nombre de algún archivo. Por ejemplo, si el usuario pregunta por el personal subrogable del contrato, y el *LLM* responde "Dicha información está presente en el Anexo VI', se procesa solamente ese archivo. Para identificar el PDF al que se refiere dentro del directorio temporal, ya que pueden haber varios archivos de tipo 'Anexo', se realiza un cálculo de distancias de similitud entre 2 cadenas. Iterativamente se calcula esa distancia entre la cadena de respuesta y cada una de las cadenas que representan el nombre de los archivos presentes. El que más *score* tenga, será al que se refiere el modelo de lenguaje. Por ejemplo, tenemos unos archivos que se pueden llamar 'AnexoIV.pdf', 'anexo_5.pdf', 'Anexo_VI.pdf', 'PCAP.pdf', 'Anuncio.pdf'. El *score* más grande se lo llevaría el 'Anexo_VI.pdf', por ser el más similar al de la respuesta que se ha mencionado más arriba. También debe ser capaz de asociar los números romanos con los números arábigos, por si existe un archivo llamado 'Anexo 6'.

3.4 Extracción de fórmulas

Existe un apartado dentro de los pliegos de cláusulas administrativas que indica los distintos criterios de puntuación automáticos que se evalúan. Estos son imprescindibles para que la entidad contratante decida con qué empresa trabajará, cuantos más puntos acumule, más encajaría con el servicio que se pide.

Suelen ser fórmulas matemáticas, acompañadas por una leyenda que explica las distintas variables que aparecen. A continuación se muestra un ejemplo:

1. Oferta económica contrato base: 50 puntos (N).

   P_{Max}= 50 x (1- ((P_{of} – P_b) / P_b))

   P_{Max} = Puntuación máxima a obtener
   P_{of} = Precio de la oferta a evaluar
   P_b= Precio de la oferta más baja

   Se considera que la oferta incluida en este apartado incurre en temeridad si fuese inferior en 15 punto porcentuales a la media de todas las ofertas recibidas.

Figura 3.12: Ejemplo sencillo de fórmula de una licitación

Se puede observar que la fórmula se puede escribir en una línea, es decir que se puede escribir fácilmente en una cadena de texto, que posteriormente se le imprima al usuario. Las librerías de lectura de PDF lo pueden identificar de manera muy fácil, como si de un texto normal se tratara. La complicación viene cuando en la fórmula aparecen fracciones o potencias, como en los siguientes ejemplos:

$$Pec = PMax * \frac{1}{N\acute{U}mOfertes} * \left[\left[\frac{import\ millor\ oferta}{import\ oferta\ (i)} + \left(N\acute{U}m\ Ofertes - 1\right) * \frac{B(i)}{Bm\grave{A}x}\right]\right]$$

Figura 3.13: Ejemplo complejo de fórmula con fracciones (extraído de PCAP original)

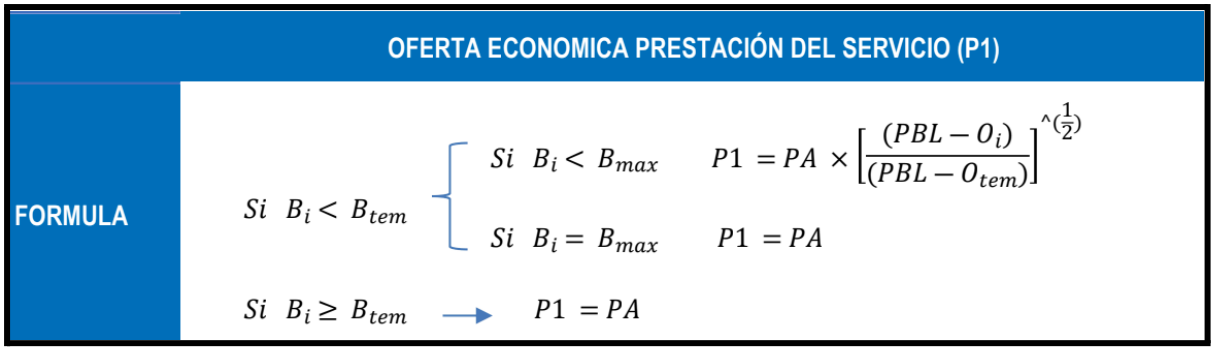


Figura 3.14: Ejemplo complejo de fórmula con fracciones y potencias

Normalmente los lectores de PDF pueden extraer los números y variables pero pierde la información sobre las operaciones. Tras muchos intentos con varias librerías se ha decidido usar un sistema de detección de objetos, en este caso especializado en identificar las fórmulas matemáticas.

Se usará la herramienta Roboflow para llevar a cabo el entrenamiento de un modelo capaz de realizar esa tarea. Los distintos pasos que se siguen se han mencionado en el apartado del estado del arte. Con Roboflow se pueden preparar los datos de entrenamiento, donde se pueden subir las imágenes que contengan fórmulas matemáticas y se deben definir los cuadros que las delimitan, como en la siguiente imagen:

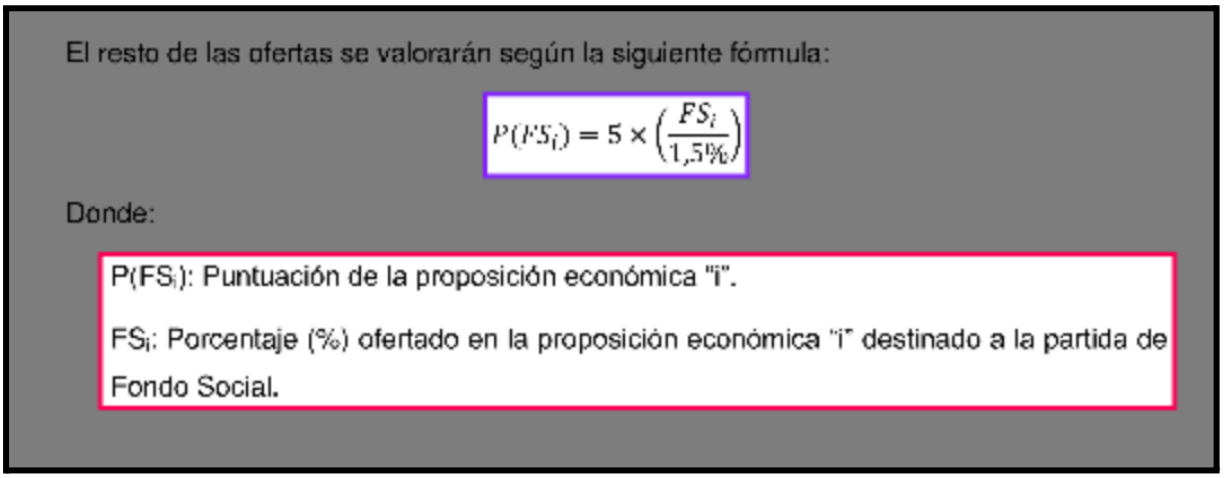


Figura 3.15: Preparación de datos de entrenamiento con Roboflow

El cuadrado morado delimita la fórmula matemática y se le atribuye la clase ‘ecuación’. Por otro lado, el cuadro rojo delimita la leyenda, y se le atribuye la clase ‘leyenda’.

Al recoger muchas imágenes con fórmulas, y marcando manualmente donde se encuentra cada una de las 2 clases anteriores, se procede a dividir en conjuntos de entrenamiento, de validación y de test. También se le aplica un preproceso para llevar todas las imágenes a la misma dimensión, y se crean nuevas imágenes a partir de las que ya hay, pero aplicando

zooms aleatorios, recortes o escalas de grises, para poder aumentar el conjunto de entrenamiento. Inicialmente había 31 imágenes etiquetadas manualmente para el conjunto de entrenamiento, y tras este paso se han conseguido 93.

El siguiente paso es ejecutar el *fine-tuning* del modelo YOLO, con los datos anteriores, especializándose para la tarea de identificar tanto fórmulas matemáticas como sus leyendas. La herramienta Roboflow también nos proporciona las métricas del entrenamiento del modelo:

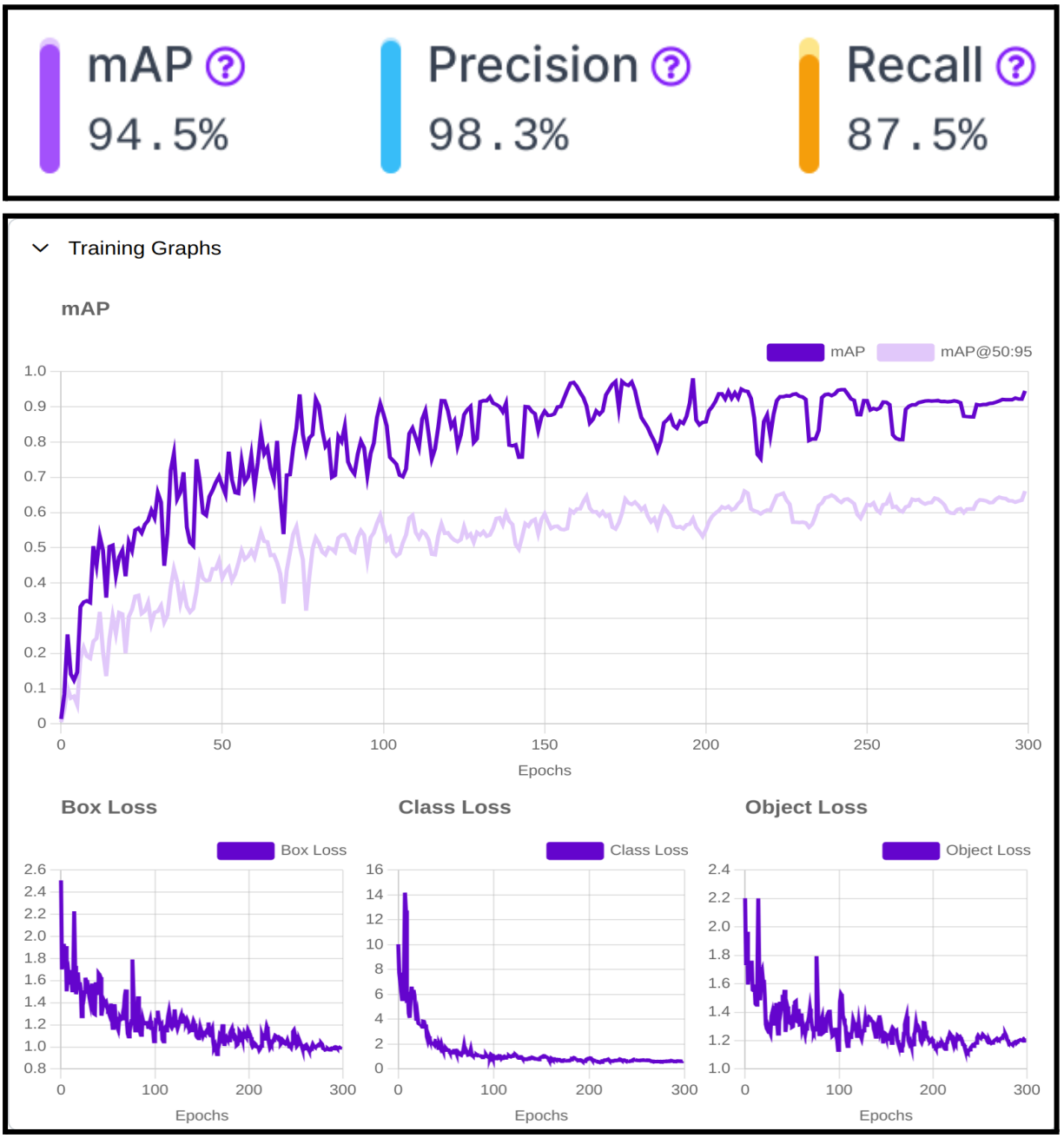


Figura 3.16: Medidas de rendimiento del modelo entrenado

Se recuerda que la precisión es la proporción de verdaderos positivos (objetos correctamente detectados) sobre el total de detecciones realizadas (la suma de verdaderos positivos y falsos positivos). El *recall* por otra parte es la proporción de verdaderos positivos sobre el total de objetos reales en el conjunto de datos (la suma de verdaderos positivos y falsos negativos)

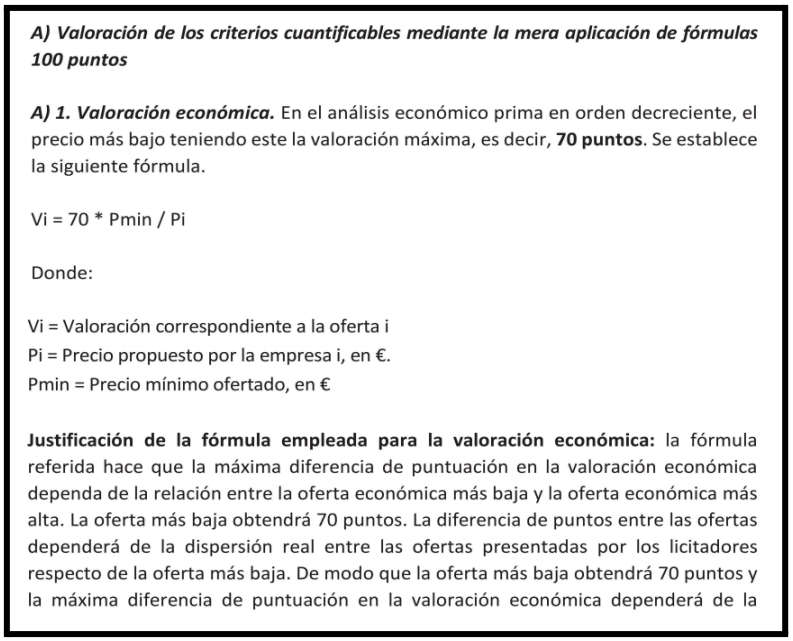
La métrica de mAP, que aparece grande en la segunda imagen, es la media de las precisiones medias para todas las clases en el conjunto de datos. Como el modelo está detectando múltiples clases de objetos (fórmulas y leyendas), se calcula la precisión media para cada clase y luego se promedian estas precisiones.

Para integrarlo en el sistema, se ha añadido un paso extra al procesar el archivo PCAP, que es el pliego, y se guardan las páginas que corresponden al apartado de los criterios de puntuación automáticos. Se guardan en formato .jpg, ya que es el admitido por el modelo de detección. Luego este será encargado de procesar la imagen e identificar los cuadros delimitadores, que serán recortados y guardados a su vez como .jpg, para que luego se le pueda imprimir al usuario.

Se ha pensado en alguna forma de introducir la librería Texify, capaz de transformar la fórmula de una imagen a un texto con formato LaTeX. Este es un sistema de composición de textos orientado a la creación de documentos escritos que presenten una alta calidad tipográfica, como artículos y libros científicos, que suelen incluir expresiones matemáticas.

Con ese enfoque se podían mostrar las fórmulas como texto normal, pero no daba siempre los resultados esperados. Por lo tanto se ha optado por mostrar directamente la imagen que contiene la fórmula, que aunque no quede muy estética, es más fiable. Para poder darle contexto al usuario sobre la imagen, no aparece solamente la fórmula, sino que incluye también un margen superior y un margen inferior para introducir de alguna forma dicha expresión.

En el informe final de una licitación quedaría tal que así:



A) Valoración de los criterios cuantificables mediante la mera aplicación de fórmulas 100 puntos

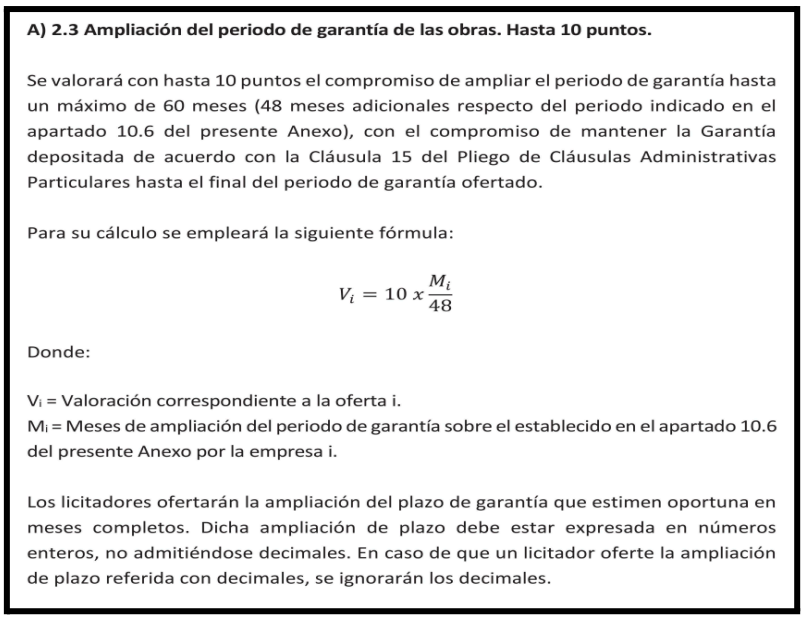
A) 1. Valoración económica. En el análisis económico prima en orden decreciente, el precio más bajo teniendo este la valoración máxima, es decir, **70 puntos**. Se establece la siguiente fórmula.

Vi = 70 * Pmin / Pi

Donde:

Vi = Valoración correspondiente a la oferta i
Pi = Precio propuesto por la empresa i, en €.
Pmin = Precio mínimo ofertado, en €

Justificación de la fórmula empleada para la valoración económica: la fórmula referida hace que la máxima diferencia de puntuación en la valoración económica dependa de la relación entre la oferta económica más baja y la oferta económica más alta. La oferta más baja obtendrá 70 puntos. La diferencia de puntos entre las ofertas dependerá de la dispersión real entre las ofertas presentadas por los licitadores respecto de la oferta más baja. De modo que la oferta más baja obtendrá 70 puntos y la máxima diferencia de puntuación en la valoración económica dependerá de la



A) 2.3 Ampliación del periodo de garantía de las obras. Hasta 10 puntos.

Se valorará con hasta 10 puntos el compromiso de ampliar el periodo de garantía hasta un máximo de 60 meses (48 meses adicionales respecto del periodo indicado en el apartado 10.6 del presente Anexo), con el compromiso de mantener la Garantía depositada de acuerdo con la Cláusula 15 del Pliego de Cláusulas Administrativas Particulares hasta el final del periodo de garantía ofertado.

Para su cálculo se empleará la siguiente fórmula:

$$V_i = 10\ x \frac{M_i}{48}$$

Donde:

V_i = Valoración correspondiente a la oferta i.
M_i = Meses de ampliación del periodo de garantía sobre el establecido en el apartado 10.6 del presente Anexo por la empresa i.

Los licitadores ofertarán la ampliación del plazo de garantía que estimen oportuna en meses completos. Dicha ampliación de plazo debe estar expresada en números enteros, no admitiéndose decimales. En caso de que un licitador oferte la ampliación de plazo referida con decimales, se ignorarán los decimales.

Figura 3.17: Criterios de puntuación de un informe generado por el sistema

3.5 Generación de informe

Ya que se han presentado los distintos componentes del informe final: texto, tablas e imágenes, el siguiente paso es juntarlos. Como se dijo al principio, hay varios apartados que tienen más importancia que otros, por los que se puede obviar que el usuario deseará saber más información acerca de estos al analizar una licitación.

Para ello habrá unos prompts predefinidos, donde para cada apartado se incluyen dos consultas: una para extraer documentos de la Vector Store, mediante la búsqueda semántica, y otra para el *LLM*, para cuando este tenga que trabajar sobre los documentos relevantes. Para los apartados especiales, que requieren un segundo paso para la extracción de tablas o imágenes de fórmulas, se usa una clase Enum 'PromptType', con los valores 'REFS' o 'IMAGES'. Para los que no necesitan este paso, tendrán el valor 'NORMAL'.

A continuación se muestra un ejemplo de cada:

```
{'type': PromptType.NORMAL,
'title': 'Organismo Contratante',
'db_query': 'Filename: Anuncio.pdf # ¿Cuál es el organismo que
solicita, emite o contrata esta licitación? Responde sólo con el
nombre',
'llm_query': 'Indica el organismo que solicita, emite o contrata
esta licitación. Responde sólo con el nombre'},
```

Figura 3.18: Ejemplo de *prompt* sin pasos extras

Hay apartados que son más fáciles de recuperar, como el que se muestra arriba. Usando unas consultas en lenguaje natural tanto para el *retriever* como para el *LLM*, se obtienen las respuestas correctas.

```
{'type': PromptType.REFS,
'title': 'Personal Subrogable',
'db_query': 'Indica, si se menciona, el anexo donde se encuentra
información sobre el personal con derechos de subrogación, el
personal subrogable, personal mínimo del contrato, personal
adscrito al servicio, la plantilla del servicio, o la plantilla
actual del contrato.',
'llm_query': 'Indica, si se menciona, el anexo donde se encuentra
información sobre el personal con derechos de subrogación, el
personal subrogable, personal mínimo del contrato, personal
adscrito al servicio, la plantilla del servicio, o la plantilla
actual del contrato.'},
```

Figura 3.19: Ejemplo de *prompt* para el apartado que incluye extracción de tablas

En el caso de los apartados que tienen un segundo paso, como el de recuperar una tabla, se le tiene que especificar al *LLM* que mencione explícitamente el nombre del anexo en el que se encuentra dicha información. Se han estudiado a fondo las licitaciones de ejemplo para identificar las palabras clave que suelen haber en estos apartados. Al mencionarlas en la pregunta, el *LLM* y el *retriever* pueden identificar de manera más fácil los pasajes que contengan la respuesta. La respuesta del modelo de lenguaje puede ser por ejemplo 'Toda la información sobre el personal subrogable de este contrato se encuentra en el anexo VI.' Esa información la extrae del archivo PCAP, que corresponde al pliego de las cláusulas. A partir de ahí se usa el método de recuperación de tablas explicado anteriormente.

```
{'type': PromptType.IMAGES,
'title': 'Criterios de Puntuación - Automáticos',
'db_query': 'Filename: PCAP.pdf Proposicion economica, criterios
de adjudicación, criterios cuantificables mediante la mera
aplicación de fórmulas, valoración de los criterios de
adjudicación, criterios evaluables de forma automática, puntos.',
'llm_query': 'Indica todos los criterios cuantificables mediante
la mera aplicación de fórmulas e indica siempre sus puntuaciones
o numero de puntos y su distribución. Da una descripción amplia
de cada uno de esos criterios, con información del texto
proporcionado. No indiques el documento o las paginas de donde se
extrae. Si existen lotes, no menciones el numero de lote.'},
```

Figura 3.20: Ejemplo de *prompt* para el apartado que incluye extracción de fórmulas matemáticas

Las consultas ‘db_query’, por las que el *retriever* buscaría los documentos relevantes, se han formulado tras analizar los documentos PDF originales. Se pueden componer de una lista de palabras clave que se suelen usar en cada apartado de los pliegos. Por ejemplo, en el caso de los criterios de puntuación automáticos, en cada licitación se mencionan de forma distinta, pero las palabras más comunes son ‘proposición económica’, ‘criterios de adjudicación’, ‘criterios cuantificables mediante fórmulas’, entre otras. A través de estas palabras, el *retriever* crearía los *embeddings* de cada una e identificaría en la Vector Store los documentos más similares a estas, y que probablemente contengan la respuesta. Esta búsqueda se realiza por la función ‘similarity search’, mencionada en apartados anteriores.

En cambio las consultas ‘llm_query’, que se le pasan al modelo de lenguaje, se deben formular como para una persona. Se le tiene que especificar los detalles que debe tener en cuenta a la hora de elaborar cada respuesta. En el ejemplo anterior de los criterios de puntuación automáticos, se le menciona al *LLM* que indique todos estos criterios, con una descripción amplia de cada uno de ellos, pero sin incluir de dónde ha extraído la información.

Los dos tipos de consulta han pasado por un proceso de *prompt engineering*, que es una técnica utilizada en el ámbito del procesamiento del lenguaje natural, que implica la creación y ajuste de los prompts. Para las consultas al *LLM*, se consigue que el modelo responda de la forma más precisa y relevante. En el caso de las consultas del *retriever,* se consigue que este sea más generalizable. Se buscan las palabras que más se mencionan en las licitaciones, para que a la hora de hacer la búsqueda semántica tenga varios ejemplos. Cada consulta se ha evaluado con distintas formulaciones, y se ha dejado la que mejor respuesta generaba.

Al subir los archivos correspondientes a una licitación iniciará automáticamente el análisis. Por cada pregunta predefinida se generará una respuesta, y estas serán mostradas en pantalla. Al acabar esta pila de prompts predefinidos, el usuario tendrá la opción de insertar una nueva pregunta, por si quiere más detalles acerca de un apartado en específico.

Las preguntas y respuestas se irán guardando como tuplas en una lista. Las respuestas de tipo texto serán de clase *string* normal, las tablas estarán en formato *html string,* y las imágenes se pasan a base64. Luego, todos los componentes se pueden pasar en formato html a un buffer, que finalmente será convertido a un archivo PDF descargable.

Capítulo 4 - Análisis de resultados

En este apartado se presentarán los distintos modelos de *embeddings* y de *LLM* que se han utilizado para el desarrollo del sistema. A continuación se mostrarán los resultados que se obtenían con el enfoque inicial, que no incluía soluciones para la extracción de tablas o de fórmulas matemáticas. Finalmente, se muestran los resultados que proporciona el estado final de la solución.

De esta forma, se puede evaluar cada combinación posible de tecnologías y decidir cuál es la mejor. Aparte de que las respuestas sean correctas, también se debe tener en cuenta el coste que suponen (por llamadas a las API) y el tiempo que tardan en ser generadas.

Modelos usados para embeddings:

Modelo	API	Precio	Evaluación MTEB
text-embedding-ada-002	OpenAI	$0.10 / 1M tokens	61.0%
text-embedding-3-small	OpenAI	$0.02 / 1M tokens	62.3%
***text-embedding-3-large** **	**OpenAI**	**$0.13 / 1M tokens**	**64.6%**

Figura 4.1: Tabla comparación modelos *embeddings*

Se puede observar que la API de OpenAI ofrece 3 modelos distintos para la generación de *embeddings*, que servirán para el *retriever* (el sistema de recuperación de documentos relevantes para una consulta). Se ha optado por elegir el último, '***text-embedding-3-large'***, ya que ha mostrado los mejores resultados, además de tener la mayor puntuación en la MTEB (Massive Text Embedding Benchmark). Esta es una *benchmark* que se dedica a medir el rendimiento de los distintos modelos que hay para esta tarea. No obstante, tiene el precio más elevado para la generación de los *embeddings* para 1 millón de *tokens*.

Modelos usados para el *LLM* generador de respuestas:

Modelo	API	Precio (*input*)	Ventana de contexto
***gpt-3.5-turbo-1106** **	**OpenAI**	**$1.00 / 1M *tokens***	**16k *tokens***
gpt-4-1106-preview	OpenAI	$10.00 / 1M *tokens*	128k *tokens*
mistral-large-latest	MistralAI	$4.00 / 1M *tokens*	32k *tokens*
command-r-plus	Cohere	$3.00 / 1M *tokens*	128k *tokens*

Figura 4.2: Tabla comparación modelos *LLM*

En cuanto al modelo que se usará en la *chain* para la generación de respuestas finales, para el informe, se ha decidido usar el GPT 3.5. Es el modelo con el precio más asequible y, aunque tenga una ventana de contexto pequeña, puede ofrecer unos buenos resultados. Es

de gran importancia el rendimiento del *retriever*, ya que interesa llenar esos 16k *tokens* de contexto solamente con información relevante. A partir de esta, el modelo es suficientemente 'inteligente' para responder de forma coherente a lo que se le pregunta.

Los demás modelos también se han utilizado para las pruebas, y hay una diferencia notable en la completitud y en el razonamiento de la respuestas, pero no se han considerado proporcionales a sus costes.

Al tener archivos muy extensos, que superan fácilmente las 100 páginas, se puede llegar a un precio muy elevado para procesar las licitaciones, por ejemplo con el modelo de GPT 4. El precio que aparece en la tabla se mide en dólares por cada millón de *tokens* que se le pasan de contexto, por lo que gastaría aproximadamente 10$ cada 7 apartados respondidos. También cabe mencionar que en tiempos de ejecución es el que más tarda, ya que al tener una ventana de contexto tan grande, de 128k *tokens*, tiene mucha más información que procesar.

La ventaja de los modelos de Cohere y de MistralAI es que pueden ser modelos *open source,* y ofrecen la posibilidad de ser desplegados en local si se tiene una máquina suficientemente potente. De esta forma el precio se reduciría considerablemente. Los modelos de OpenAI no ofrecen esta opción.

Tras muchas pruebas, el modelo ***'gpt-3.5-turbo-1106'*** ha demostrado ser el más viable en cuanto a respuestas que genera, precio y tiempos de ejecución. A continuación se muestran los resultados obtenidos con 2 versiones del sistema al analizar una misma licitación, utilizando en los 2 casos este modelo de *LLM* y el modelo de *embeddings* '***text-embedding-3-large'***.

Enfoque 1

Esta es la versión inicial del sistema, que no tenía en cuenta la dificultad de recuperar la información estructurada de las tablas ni de las expresiones matemáticas complejas. Los PDF se analizan con una herramienta básica de lectura 'PdfReader', y se extrae todo el texto en 'crudo', sin formato. Se va a mostrar una tabla que recoge la información sobre algunos apartados del informe que se genera sobre un ejemplo de licitación.

Ejemplo

Apartado	**Valoración**	**Nr. Documentos**	**Precisión**	***Recall***
Organismo Contratante	Correcto	10	0.4	1
Plazo de Presentación	Correcto	10	0.2	1
Localización	Incorrecto	10	0.1	1
Objeto del Contrato	Correcto	10	0.2	1
Duración	Correcto	10	0.1	1
Valor Estimado del Contrato	Correcto	10	0.2	1
Solvencia económica y financiera	Parcialmente correcto	9	0.44	0.88
Solvencia técnica o profesional	Parcialmente correcto	9	0.44	1
Personal Subrogable	Incorrecto	10	0.2	1
Criterios de Puntuación - Automáticos	Parcialmente correcto	8	0.625	1

Figura 4.3: Resultados ejecución de ejemplo (Enfoque inicial)

En la columna 'Valoración' se indica 'correcto' cuando la respuesta del sistema contiene el 100% de la información correcta, 'incorrecto' cuando contiene menos del 50% y 'parcialmente correcto' cuando contiene entre 50% y 90%. Hay apartados como 'Localización' y 'Personal Subrogable' en los que el *LLM* ha fallado al generar la respuesta. En el primero ha respondido con la dirección equivocada, aunque haya acertado el código postal y la provincia, y en el segundo se ha limitado a mencionar el documento anexo en el

que se encuentra información sobre este personal subrogable, pero no lo ha detallado porque este viene en formato de tabla y no es capaz de extraer su información.

En la columna de 'Nr. documentos' se indica el número de documentos que son retornados por el *retriever*, como relevantes para la consulta del usuario. Se recuerda que hay un método para mantener el número de *tokens* dentro del límite de la ventana de contexto del *LLM*, y por ello hay algunos apartados en los que se recuperan menos documentos. Las solvencias y los criterios de puntuación son apartados muy extensos, por lo que contienen muchos *tokens*, y es necesario limitarlos.

Para cada apartado, se analizan los documentos que retorna el *retriever* al *LLM* para que este genere la respuesta. Si estos son los correctos, la respuesta tiene más probabilidad de ser la correcta. Por lo contrario, el *LLM* no tendrá el contexto suficiente como para responder al usuario, y puede dar una respuesta parcial o directamente incorrecta.

Para calcular las medidas de 'precisión' y '*recall'*, que indican el rendimiento del sistema, se han utilizado las fórmulas mencionadas en el capítulo del Estado del arte:

$$\text{Precisión} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ recuperados\}|}$$

$$\text{Recall} = \frac{|\{documentos\ relevantes\} \cap \{documentos\ recuperados\}|}{|\{documentos\ relevantes\}|}$$

Como ejemplo se puede tomar la respuesta al apartado 'Organismo contratante', que de 10 documentos recuperados, se menciona el organismo en 4 de ellos. De todos los documentos existentes (retornados o no), también se menciona el organismo solo en 4. Por lo que obtenemos una precisión de 4/10, y un *recall* de 4/4.

La precisión siempre será más baja, porque se le ha definido al *retriever* un valor *k=10*, por lo que siempre va a devolver el mayor número posible de documentos (hasta 10), y no se va a limitar a los estrictamente necesarios. Siempre habrá documentos que no contengan información sobre el apartado, pero estos tendrán una menor relevancia en el ranking del *retriever*.

El *recall* en cambio será mayoritariamente 1, ya que suele recuperar todos los documentos relevantes. No obstante, para el apartado de la 'Solvencia económica y financiera', no se han recuperado todos. Por ello se hará un cálculo de 8 documentos relevantes recuperados sobre los 9 relevantes que existen, 8/9.

En el caso del personal subrogable, existen 2 documentos relevantes. Uno de ellos, el archivo PCAP (3).pdf', es el que hace referencia al archivo anexo 'Personal a subrogar.pdf'. Este último es el que contiene una tabla pero no es capaz de mostrarla de ninguna forma en el informe final.

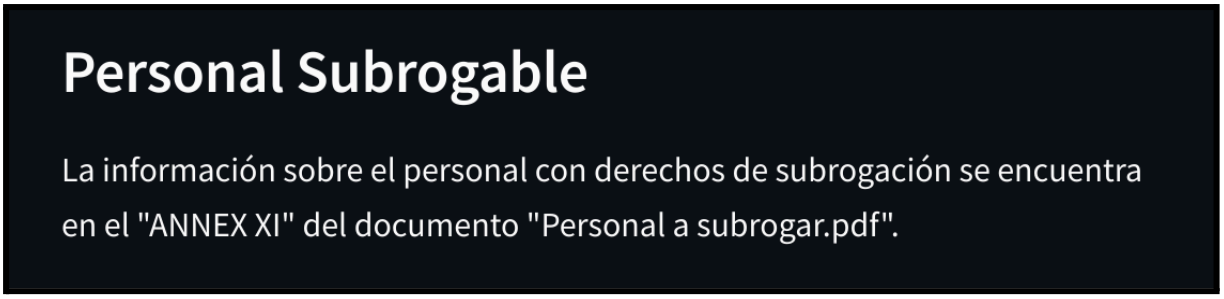


Figura 4.4: Resultado 'Personal Subrogable' (Enfoque inicial)

En cuanto a las fórmulas matemáticas que se muestran, podemos observar que ha sido capaz de extraer las expresiones sencillas como:

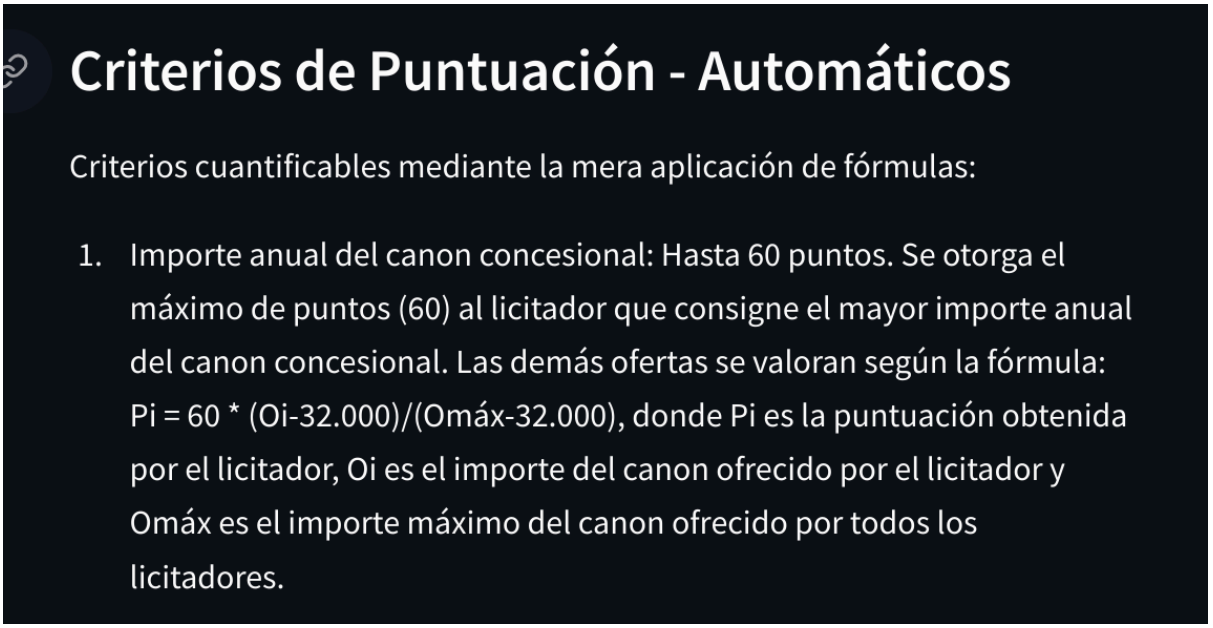


Figura 4.5: Resultado 'Criterios puntuación' (Enfoque inicial)

La expresión anterior se trata de una expresión que puede ser convertida a formato *string* de forma fácil, y se puede imprimir en el output. En cambio, también hay fórmulas como la siguiente:

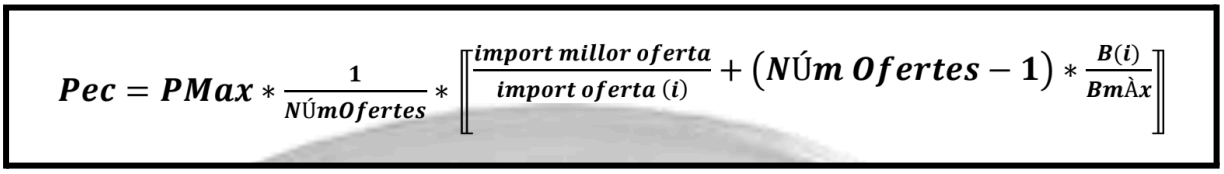


Figura 4.6: Ejemplo de fórmula matemática original

La librería de lectura de PDF no es capaz de identificar correctamente todas las líneas de fracciones y la transcribe como:

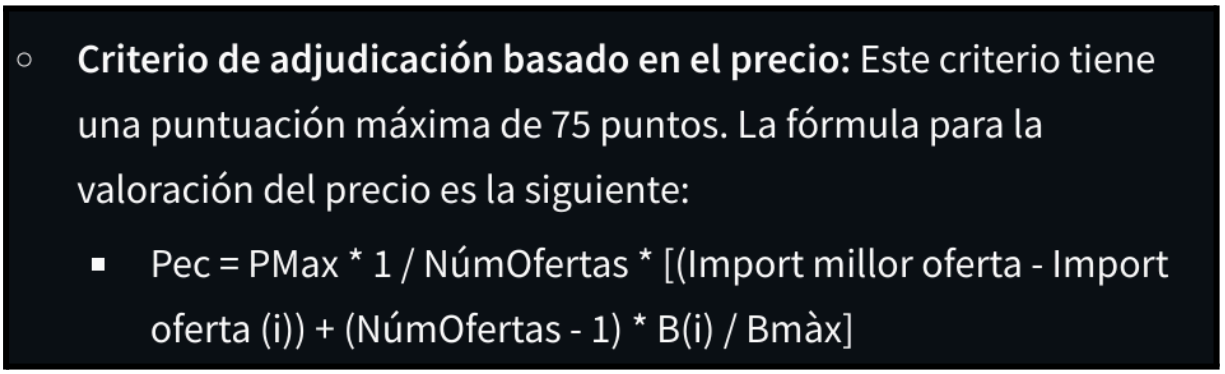


Figura 4.7: Ejemplo de fórmula matemática compleja (Enfoque inicial)

Para solucionar los problemas de las tablas y las fórmulas, se ha trabajado sobre un segundo enfoque, que tras probar muchas formas de integrarlo, ha llegado a una solución final.

Enfoque 2 - Solución final

Esta es la versión del sistema que incluye las soluciones *ad-hoc* para los mayores inconvenientes del primer enfoque: fallos en extracción de tablas y fórmulas matemáticas. Tras introducir la lectura de PDF con OCR y el modelo de detección de objetos, se ha vuelto a ejecutar el sistema con el mismo ejemplo de licitación. Se han obtenido los siguientes resultados:

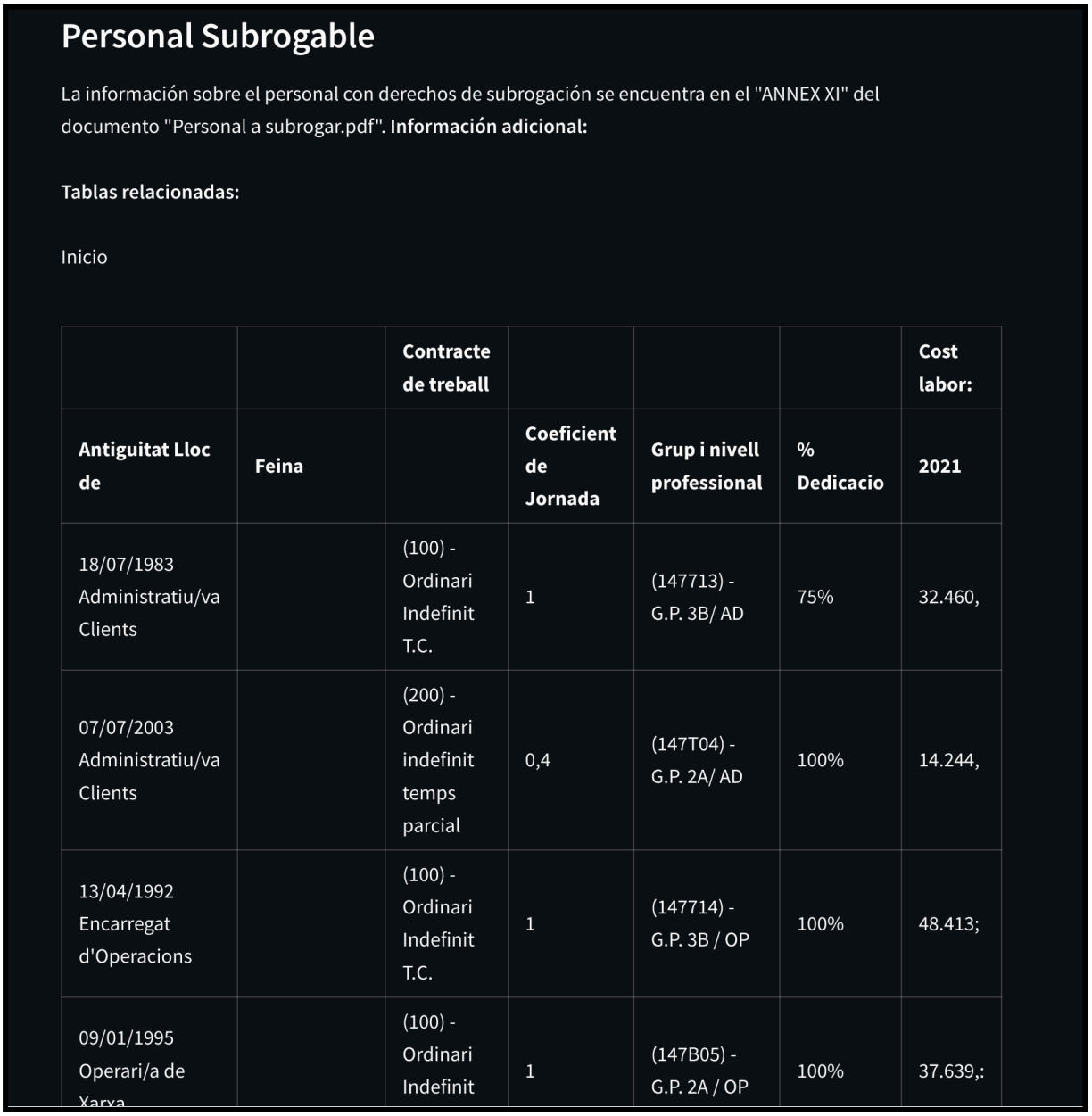
Apartado	**Valoración**	**Nr. Documentos**	**Precisión**	***Recall***
Organismo Contratante	Correcto	10	0.4	1
Plazo de Presentación	Correcto	10	0.2	1
Localización	Incorrecto	10	0.1	1
Objeto del Contrato	Correcto	10	0.2	1
Duración	Correcto	10	0.1	1
Valor Estimado del Contrato	Correcto	10	0.2	1
Solvencia económica y financiera	Parcialmente correcto	9	0.44	0.88
Solvencia técnica o profesional	Parcialmente Correcto	9	0.44	1
Personal Subrogable	Correcto	10	0.2	1
Criterios de Puntuación - Automáticos	Correcto	8	0.625	1

Figura 4.8: Resultados ejecución de ejemplo (Enfoque final)

Se observa que los valores de los números de documentos recuperados y de las medidas de rendimiento siguen siendo los mismos. Se debe a que el sistema es consistente, no genera respuestas distintas para un mismo *input* (se analiza la misma licitación que en la tabla anterior). Esto se consigue mediante el *prompt engineering*, el *system template* y por los parámetros de temperatura y de *seed* del *LLM*.

La diferencia está en los resultados del ‘personal subrogable’ y de los ‘criterios de puntuación’. Esta vez se aplica un segundo paso para cada uno de los apartados.

Para el primero, al recibir la respuesta: ‘La información sobre el personal con derechos de subrogación se encuentra en el "ANNEX XI" del documento "Personal a subrogar.pdf"’, se lanza un segundo proceso que consta en analizar dicho archivo, y es capaz de extraer la tabla y plotearla en formato *html*:



Personal Subrogable

La información sobre el personal con derechos de subrogación se encuentra en el "ANNEX XI" del documento "Personal a subrogar.pdf". **Información adicional:**

Tablas relacionadas:

Inicio

		Contracte de treball				**Cost labor:**
Antiguitat Lloc de	**Feina**		**Coeficient de Jornada**	**Grup i nivell professional**	**% Dedicacio**	**2021**
18/07/1983 Administratiu/va Clients		(100) - Ordinari Indefinit T.C.	1	(147713) - G.P. 3B/ AD	75%	32.460,
07/07/2003 Administratiu/va Clients		(200) - Ordinari indefinit temps parcial	0,4	(147T04) - G.P. 2A/ AD	100%	14.244,
13/04/1992 Encarregat d'Operacions		(100) - Ordinari Indefinit T.C.	1	(147714) - G.P. 3B / OP	100%	48.413;
09/01/1995 Operari/a de Xarxa		(100) - Ordinari Indefinit	1	(147B05) - G.P. 2A / OP	100%	37.639,:

Figura 4.9: Resultado ‘Personal Subrogable’ (Enfoque final)

El *output* no es muy estético y tampoco está estructurado de manera perfecta, pero puede aportar la información necesaria al usuario.

Para el problema con las expresiones matemáticas complejas se ha añadido el paso intermedio de un modelo de detección de objetos entrenado específicamente para fórmulas. Para la fórmula compleja mostrada anteriormente se ha obtenido el siguiente resultado:

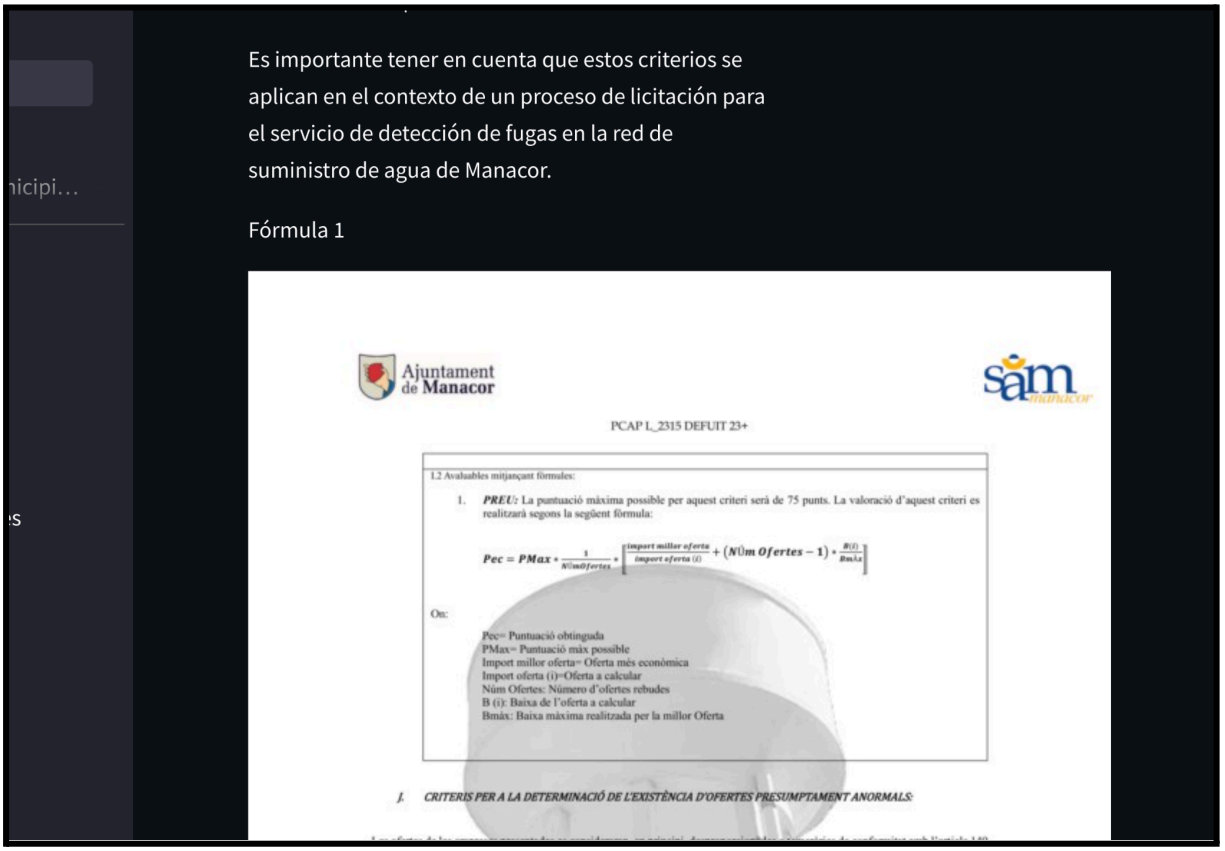


Figura 4.10: Ejemplo de fórmula matemática compleja (Enfoque final)

De la misma manera que con las tablas, la forma de mostrar las fórmulas no es la más estética, pero tras probar integrar el formato LaTeX no se ha llegado a una solución apta para todas las licitaciones, ya que cada una tenía un formato distinto. Mostrarlas en forma de imagen era la solución más generalizable.

Capítulo 5 - Conclusión

El objetivo principal era conseguir un generador automático de informes que replique a los que generaría un humano. Tras analizar los resultados obtenidos, uno puede concluir que el sistema puede funcionar bastante bien. El informe devuelto al usuario incluye información interesante acerca de algunos apartados. No obstante, la licitación procesada era muy 'optimista'. Tenía un formato sencillo que favorece al sistema, y aun así no ha sido capaz de extraer toda la información relevante.

Analizando otras licitaciones, se observa que muchas de las respuestas no alcanzan el nivel esperado, y pueden ser incompletas o incorrectas. El objetivo era conseguir un sistema que devolviera un informe 100% fiable, sin margen de error, como lo haría una persona. Tras múltiples refinamientos sobre los *prompts,* para que el modelo de lenguaje mejore su respuesta, no se ha conseguido esa perfección.

Existen respuestas del informe que pueden ser críticas a la hora de que una empresa elija si les conviene un contrato o no. Si no se puede fiar de que el sistema le brinde siempre el 100% de la información que necesita, no puede dar el paso a que sea un proceso automatizado. No obstante, se le puede encontrar utilidad al analizar otro tipo de archivos, que no sean licitaciones, y que admitan un margen de error en cuanto a la correctitud de las respuestas.

Para la aplicación que se ha desarrollado en este trabajo no existe una solución única. Se pueden probar otras tecnologías aparte de las que han sido mencionadas, se pueden probar distintos parámetros en los métodos utilizados y los *prompts* se pueden reformular de muchas formas distintas. Hay margen de mejora, pero de momento se excluye la posibilidad de conseguir un sistema con 100% de eficacia.

La inteligencia artificial está en constante desarrollo y progreso, no obstante, todavía no ha llegado a un nivel de perfección absoluta. Se han conseguido ya sistemas que hace unos años eran impensables, como ChatGPT, Vehículos Autónomos, Asistentes Virtuales, entre otros, pero no en todos los ámbitos llegan a la capacidad de reemplazar a un humano.

En un futuro, conforme vayan mejorando los modelos de lenguaje y los sistemas de recuperación de información, se podría intentar mejorar el sistema. Los LLM serán más 'inteligentes', posiblemente con una mayor ventana de contexto, y serán capaces de generar una respuesta más fiable. A su vez, los *retrievers* pueden mejorar a la vez que se mejora la generación de *embeddings*, pudiendo extraer mejor los pasajes de información que sean relevantes para una consulta.

Bibliografía

[1] A., Enrique (2018). Detección de objetos con YOLO: implementaciones y como usarlas. URL: https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246

[2] Bush, Vannevar (1945). As We May Think. Atlantic Monthly, 176:101–108.

[3] Calvo, Diego (2017). Red Neuronal Convolucional CNN. URL: https://www.diegocalvo.es/red-neuronal-convolucional/

[4] Calzone, Ottavio (2022). An Intuitive Explanation of LSTM. URL: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c

[5] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. URL: https://arxiv.org/abs/1810.04805

[6] D. K. Harman (1993). Overview of the first Text REtrieval Conference (TREC-1). In Proceedings of the First Text REtrieval Conference (TREC-1), pages 1–20. NIST Special Publication 500-207.

[7] Draelos, Rachel (2019). Measuring Performance: The Confusion Matrix. URL: https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/

[8] Fernández Jauregui, Ander (2020). Qué son y cómo crear una red neuronal convolucional con Keras. URL: https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/

[9] Hosni, Youssef (2023). Hands-On LangChain for LLM Applications Development: Documents Splitting [Part 1]. URL: https://youssefh.substack.com/p/hands-on-langchain-for-llm-applications-2d6

[10] Ilkin Serengil, Sefik (2020). A Gentle Introduction to ROC Curve and AUC in Machine Learning. URL: https://sefiks.com/2020/12/10/a-gentle-introduction-to-roc-curve-and-auc/

[11] Keita, Zoumana (2024). Explicación de la detección de objetos YOLO. URL: https://www.datacamp.com/es/blog/yolo-object-detection-explained

[12] McCracken, Cole (2018). Deep Neural Networks: Choosing a Learning Rate. URL: https://medium.com/@colemccracken/deep-neural-networks-choosing-a-learning-rate-172b97ef459

[13] Mishra, Onkar (2023). Using langchain for Question Answering on Own Data. URL: https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed

[14] O'shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458. URL: https://arxiv.org/abs/1511.08458

[15] Powers, D. M. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061. URL: https://arxiv.org/abs/2010.16061

[16] Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. URL: https://paperswithcode.com/paper/improving-language-understanding-by

[17] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788). URL: https://arxiv.org/abs/1506.02640

[18] Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. arXiv preprint arXiv:1912.05911. URL: https://arxiv.org/abs/1912.05911

[19] S., Juan (2020). Redes Neuronales Recurrentes. URL: https://juansensio.com/blog/034_rnn_intro

[20] Sepp Hochreiter, Jürgen Schmidhuber (1997). Long Short-Term Memory. Neural Comput, 9 (8): 1735–1780.

[21] Shah, Deval (2023). Intersection over Union (IoU): Definition, Calculation, Code. URL: https://www.v7labs.com/blog/intersection-over-union-guide

[22] Singhal, Amit (2001). Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35-43.

[23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30. URL: https://arxiv.org/abs/1706.03762

OBJETIVOS DE DESARROLLO SOSTENIBLE (Anexo I)

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS1. **Fin de la pobreza**				**X**
ODS2. **Hambre cero**				**X**
ODS3. **Salud y bienestar**				**X**
ODS4. **Educación de calidad**				**X**
ODS5. **Igualdad de género**				**X**
ODS6. **Agua limpia y saneamiento**				**X**
ODS7. **Energía asequible y no contaminante**				**X**
ODS8. **Trabajo decente y crecimiento económico**				**X**
ODS9. **Industria, innovación e infraestructura**	**X**			
ODS10. **Reducción de las desigualdades**				**X**
ODS11. **Ciudades y comunidades sostenibles**				**X**
ODS12. **Producción y consumo responsable**				**X**
ODS13. **Acción por el clima**	**X**			
ODS14. **Vida submarina**				**X**
ODS15. **Vida de ecosistema terrestres**				**X**
ODS16. **Paz, justicia e instituciones sólidas**				**X**
ODS17. **Alianzas para lograr objetivos**				**X**

Dado el contenido del presente trabajo, se considera que se han cumplido los objetivos 'ODS9. **Industria, innovación e infraestructura**' y 'ODS13. **Acción por el clima**'.

El primero, ODS9, se cumple al intentar automatizar un proceso que hasta ahora solo se ha llevado a cabo por personas. Estas pasan muchas horas analizando pliegos de cláusulas administrativas de largas extensiones, que pueden superar las 100 páginas, para luego realizar un informe a mano con los apartados más importantes. Se ha probado integrar un sistema de *Question Answering*, una tarea ya existente en el campo del procesamiento de lenguaje natural, pero no llega a las expectativas del usuario.

El segundo objetivo, ODS13, se consigue al usar como modelo de detección de objetos uno ya preentrenado, YOLO. A este se le aplica un proceso de *fine-tuning* que lo especializa para el tipo de datos que existen en este proyecto, las fórmulas matemáticas. Al no entrenar un modelo desde 0, no ha habido necesidad de invertir los recursos computacionales que una tarea así requiere, por lo que se reduce el impacto de la huella de carbono.