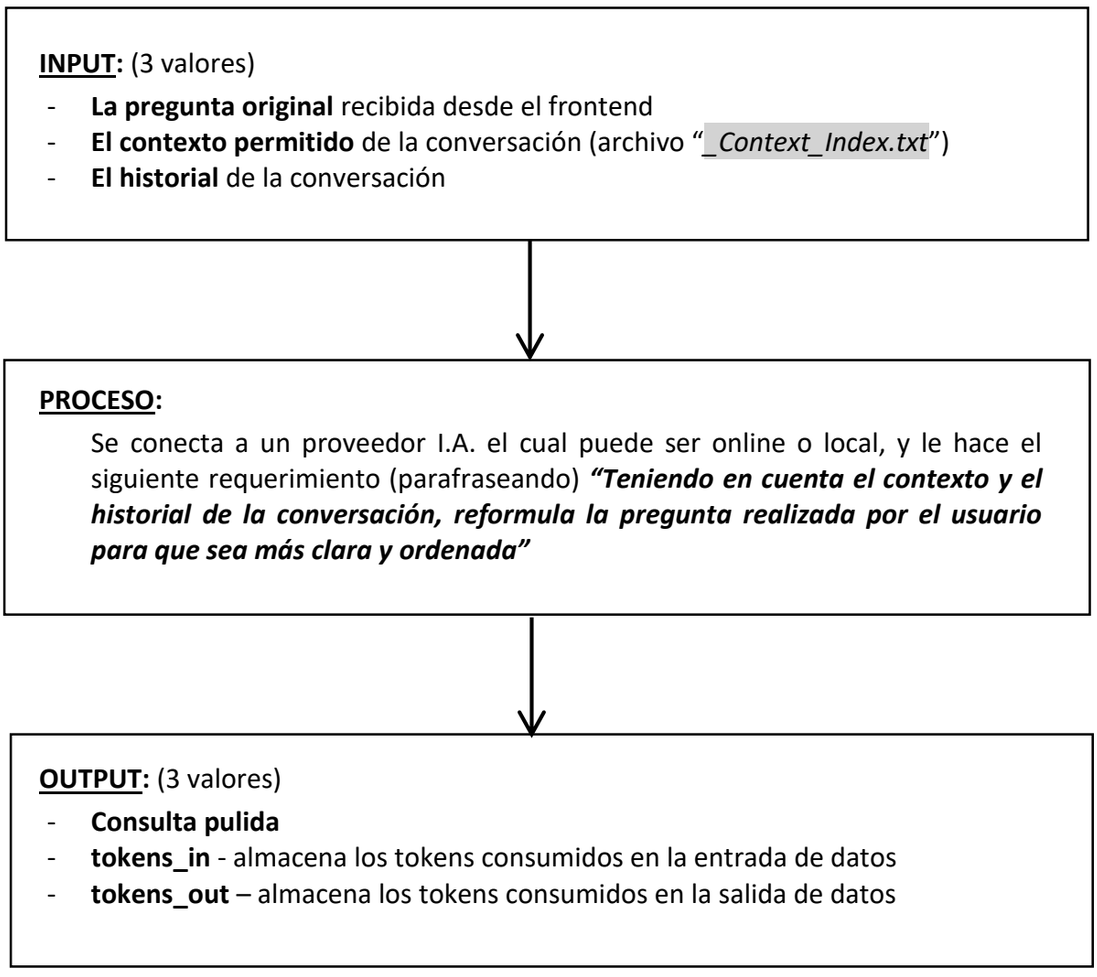
Sección 5 – Flujo Principal del Algoritmo en smartFunctions.js

Esta sección describe el funcionamiento de las diferentes funciones inteligentes que se hallan en el archivo smartFunctions.js.

1. async function pulirPregunta()

```
async function pulirPregunta(preguntaOriginal, contextoPermitido, historial)
return { consultaPulida, tokens_in, tokens_out }
```

Descripción: Esta smart function tiene como objetivo tomar la pregunta imperfecta que el usuario incorpora a la conversación, y entregar como retorno un **prompt perfecto**, el cual es necesario para cumplir con una pregunta comprensible, dentro del contexto de la conversación y su historial.



Esta función cumple múltiples propósitos: **(1)** Por una parte permite que el usuario realice exclamaciones y comentarios informales, sin que el flujo de la conversación se vea afectado. **(2)** Por otra parte, permite que el contexto y el historial estén correctamente incorporados al prompt, de manera que no será necesario invocar nuevamente el historial en los múltiples próximos requerimientos del algoritmo, ahorrando por lo tanto muchos tokens y evitando procesamiento redundante.

2. async function identificarArchivos()

async function identificarArchivos(preguntaPulida, documentIndex, maxChunks)
return { archivosRelevantes, tokens_in, tokens_out }

Descripción: Con la consulta optimizada, el algoritmo busca los archivos más relevantes en el índice de documentos. Esta Smart-Function devolverá el listado de chunks donde habrá de buscarse la respuesta a la consulta.

INPUT: (3 valores)

- Se entrega la pregunta Pulida, el **prompt perfecto** como consulta
- Se accede al índice de documentos disponible en "_Document_Index.txt"
- Se entrega el parámetro MAX_CHUNKS sobre los cuales se buscará.

PROCESO:

Se conecta a un proveedor I.A. el cual puede ser online o local, y le hace el siguiente requerimiento (parafraseando) ***"El usuario ha hecho esta consulta, y aquí está el índice del contenido del documento en sus diferentes chunks. Teniendo en cuenta el índice ¿a cuáles chunks irías a buscar la información para contestar la consulta del usuario?"***

OUTPUT: (3 valores)

- **Retorna archivosRelevantes**, un string conteniendo los nombres de los archivos relevantes, sobre los cuales iría a buscar la respuesta a la consulta.
- **tokens_in** - almacena los tokens consumidos en la entrada de datos
- **tokens_out** – almacena los tokens consumidos en la salida de datos

3. async function construirRespuesta()

async function construirRespuesta(archivos, textoConsulta, maxRespuestas)
return { respuestaFinal, chunksExaminados, cantRespPositivas, tokens_in, tokens_out };

Descripción: Esta función realiza la búsqueda de respuestas a la consulta realizada, dentro de cada uno de los chunks listados en la función de identificarArchivos(). Se combina la pregunta original con la versión refinada (prompt perfecto) para enriquecer la búsqueda. La resultante puede ser una o más respuestas válidas a la consulta.

INPUT: (3 valores)

- **archivos** es el listado de chunks donde se buscarán las respuestas
- **textoConsulta** es la concatenación de la pregunta original + consulta pulida
- **maxRespuestas** es el valor MAX_ANSWERS de la configuración

PROCESO:

Se conecta a un proveedor I.A. el cual puede ser online o local, y le hace el siguiente requerimiento (parafraseando) ***"Responde preguntas basadas estrictamente en el resultado del documento. Menciona también el índice y/o Sección y/o Artículo legal y/o Capítulo y versículo y agrega la cita textual. No infieras, no agregues, no hagas suposiciones. Si no encuentras la respuesta responde ´-´ un solo carácter."***

PROCESO:

Este proceso de consulta se repite tantas veces como chunks fueron seleccionados para la búsqueda. El algoritmo se detiene prematuramente si alcanza el número de **maxRespuestas** positivas dentro del documento.

Las respuestas positivas se van concatenando (sumando) para entregar una respuesta final, compuesta por todas las respuestas positivas encontradas.

OUTPUT: (5 valores)

- **respuestaFinal** la concatenación de todas las respuestas positivas encontradas
- **chunksExaminados** el número de archivos-trozos efectivamente consultados
- **cantRespPositivas** respuestas positivas encontradas
- **tokens_in** - almacena los tokens consumidos en la entrada de datos
- **tokens_out** – almacena los tokens consumidos en la salida de datos

Esta Smart-Function es el algoritmo más largo que se ejecuta, realizando múltiples consultas a la I.A.; las cuales sin embargo no son exigentes, ni en procesamiento ni en memoria, dado que no hay embeddings involucrados en toda la operación, en ninguno de los pasos.

4. async function pulirRespuesta()

```
async function pulirRespuesta(pregunta, contextoPermitido, respuestaFinal)
return { respuestaPulida, tokens_in, tokens_out }
```

Descripción: En virtud de que el algoritmo anterior construirRespuesta() puede contener varias respuestas válidas a la misma consulta, esta función tiene como fin entregar una única respuesta clara y ordenada.

Del mismo modo que si un estudiante hubiera entrado en una biblioteca con una pregunta anotada en un papel, y luego de haber consultado varios textos encuentra varias respuestas válidas. Ahora se dispone a efectuar un resumen de todas ellas para entregar una única respuesta clara y ordenada. Eso es lo que este último algoritmo hace.

INPUT: (3 valores)

- **pregunta** ingresa a este algoritmo la pregunta pulida (únicamente)
- **contextoPermitido** el algoritmo se asegura que no se haya ingresado una pregunta que esté fuera del contexto permitido de la conversación.
- **respuestaFinal** el conjunto de respuestas positivas concatenadas.

PROCESO:

Se conecta a un proveedor I.A. el cual puede ser online o local, y le hace el siguiente requerimiento (parafraseando) ***"Organiza y presenta las respuestas de manera armoniosa como una sola respuesta, Usa Formato Markdown. Usa negritas para resaltar ideas principales. Usa viñetas y saltos de línea para mejorar la legibilidad. Solo puedes responder en base al contexto permitido. Organiza esta respuesta y no agregues ningún dato fuera de lo que está aquí. No autocomplete, no agregues opinión ni suposiciones."***

OUTPUT: (3 valores)

- **respuestaPulida** el resultado final, la respuesta final que se mostrará en la consola, en el frontend.
- **tokens_in** - almacena los tokens consumidos en la entrada de datos
- **tokens_out** – almacena los tokens consumidos en la salida de datos

Conclusión Final:

Estas son las cuatro Smart-Functions que hacen posible que este algoritmo pueda leer documentos ilimitados, no necesitando modelos LLM's de gran tamaño, no requiriendo embeddings ni entrenamiento, entregando respuestas precisas y sin alucinaciones, a un precio mínimo.

El secreto de todo el código se basa en tres fundamentos base:

(1) **El paradigma de la Curva de Conceptos**: que establece que un conocimiento o un documento no deben expresarse como un vector inserto en un espacio de cientos de dimensiones, sino como un conjunto de conceptos simples interconectados.

(2) **El método Concept Curve Embeddings Indexation:** una nueva manera de indexar los documentos, que no se basa en comprimir las ideas formando vectores de miles de dimensiones, sino en indexar los documentos como un conjunto de conceptos simples interconectados.

(3) **Las Smart-Functions:** son funciones que requiere Inteligencia Artificial, las cuales a diferencia de las function-calling de OpenAI; éstas no son dependientes de ningún proveedor y pueden ejecutarse en modelos pequeños y localmente.

El algoritmo **es de muy bajo costo monetario** porque no precisa de los modelos LLM's de avanzada para poder funcionar, sino que funciona con los modelos más livianos.

El algoritmo **no es exigente en procesamiento computacional** porque no utiliza embeddings, los cuales requieren de capacidad de cómputo para comprimir ideas en vectores de miles de dimensiones y compararlas una por una con otros vectores de miles de parámetros.

El algoritmo **no es exigente en almacenamiento** de datos porque no almacena vectores con miles de parámetros, sino que solo se almacenan simples conjuntos de palabras, las cuales son conceptos, que representan finalmente textos.

Para mayor información, consulte con el Agente CC, una de las personalidades de ChatGPT diseñada para explicar este método: https://tinyurl.com/agente-cc

En la última sección del Manual se enseñará el método para realizar la indexación según el Concept Curve Embeddings Indexation. En el directorio /data donde se encuentran los trozos de documento (chunks) se encuentra también el archivo *_Document_Index.txt* en el cual podrá ver cómo se realiza la indexación según el CC EI. Más información será publicada en el canal de youtube

https://www.youtube.com/@Agente_Concept_Curve

Eso es todo. Muchas Gracias.

Daniel Bistman