TUIA NLP 2025
TRABAJO PRÁCTICO FINAL

Pautas generales:

- **Modalidad**: El trabajo deberá ser realizado de forma individual.
- **Entorno**: El proyecto debe desarrollarse íntegramente en un cuaderno de Google Colab.
- **Entregables**: Se debe entregar un informe en PDF y el cuaderno Jupyter (.ipynb) ejecutado y sin errores. Ambos deben estar en un repositorio de Git (GitHub/GitLab) compartido con los docentes.
    - jpmanson@gmail.com
    - alan.geary.b@gmail.com
    - constantinoferrucci@gmail.com
- **Fecha estimada límite de entrega**: Domingo 7 de Diciembre a las 23:59.

Objetivo del trabajo práctico

Desarrollar un asistente virtual especializado en información de una empresa de electrodomésticos que responde preguntas como por ejemplo:

Información sobre usos de productos:
- "¿Cómo uso mi licuadora para hacer smoothies?"

Información sobre precios de productos:
- "¿Cuáles son las licuadoras de menos de $200?"

Información sobre reseñas de usuarios:
- "¿Qué opinan los usuarios de esta cafetera?"

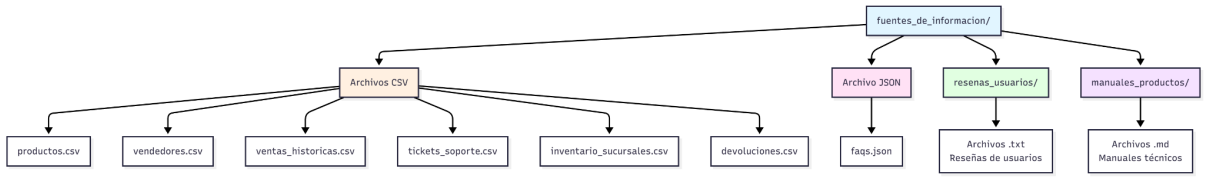
Recomendación sobre productos:
- “Quiero una licuadora con buenas reseñas”

Respuestas a preguntas frecuentes:
- “¿Qué voltaje requiere el rallador digital eléctrico?”

Respuesta sobre productos:
- “¿Qué productos están relacionados con la categoría Cocina?”

Fuente de información a utilizar

fuentes_de_informacion



Nota: Para el primer ejercicio no se utilizarán TODOS los archivos.

Ejercicio 1 - RAG

Requisitos y pasos:

- A partir de los datos dados generar tres tipos de fuentes de datos y definir/justificar su contenido:
    - **Vectorial**
    - **Tabular**
    - **Grafos**

- Para la **base de datos vectorial** es necesario:
    - Definir un modelo de base de datos vectorial (ChromaDB o FAISS por ejemplo).
    - Definir un modelo de embedding que maneje información en español.
    - Definir un Text Splitter con parámetros razonables para nuestra fuente de información.
    - Preparar una interfaz para acceder mediante una función en donde le pasaremos argumentos (consulta y filtros) y nos devolverá K fragmentos más parecidos a esa consulta.

- Para la **base de datos tabular**:
    - Cargar la información en Dataframes de Pandas o Tablas de SQL.
    - Recopilar información de importancia, como valores únicos (en caso de campos categóricos), mínimos o máximos (en campos enteros) y otra información relevante, con el propósito de utilizarlos en la construcción de un string con formato para que el LLM pueda filtrar en el set de datos.
    - Utilizar un modelo de lenguaje con instrucciones de sistema ajustado a la información de nuestro set de datos (NO PASAR TODO EL SET DE DATOS ENTEROS, SOLO INFORMACIÓN EXTRAÍDA EN EL PUNTO ANTERIOR) y solicitar al modelo de lenguaje que nos de los filtros correspondientes para buscar la respuesta a la consulta que necesitamos.
    - Utilizar esta respuesta obtenida para filtrar nuestro set de datos y devolver la respuesta
    - Al igual que la fuente de datos anterior, precisamos desarrollar una interfaz para acceder mediante una función en donde solamente le pasaremos filtros los cuales serán definidos por un modelo de lenguaje experto en código.

- Crear una **base de datos de grafos** con los siguientes requisitos:
    - Importar los datos obtenidos de relaciones en un dataframe y preparar para su inserción en la base de datos de grafos.
    - Para la base de datos de grafos es obligatorio usar una que se pueda consultar con Cypher.
    - Al igual que el punto anterior, se precisa implementar un modelo de lenguaje que convierta la consulta del lenguaje natural a consulta Cypher.
    - Utilizar esta respuesta obtenida para filtrar nuestro set de datos y devolver la respuesta

    - Al igual que la fuente de datos anterior, precisamos desarrollar una interfaz para acceder mediante una función en donde solamente le pasaremos la consulta la cuál será definida por un modelo de lenguaje experto en código.

- Desarrollar un **Clasificador de Intención Avanzado**:
    - Tomar:
        - Clasificador entrenado propio con preguntas sintéticas sobre cada fuente de datos.
        - Clasificador basado en LLM con Few-Shot Prompting.
    - Compara ambos clasificadores (el entrenado y el basado en LLM), justificando cuál es el mejor usando métricas de clasificación.

- Definir un **Pipeline de Recuperación (Retrieval)**:
    - Para consultas en bases de datos semánticas:
        - Implementa una búsqueda híbrida que combine la búsqueda semántica con una búsqueda por palabras clave (ej: BM25).
        - Aplica un mecanismo de ReRank sobre los resultados para mejorar la relevancia de los chunks recuperados.
    - Para consultas en bases de datos de grafos y datos tabulares (como se indicó anteriormente):
        - Las consultas a estas fuentes deben ser dinámicas. No está permitido pasar la tabla o el grafo completo al contexto.
        - Crea funciones que, a partir del prompt del usuario, generen una query SQL o un filtro de Pandas para la tabla, y una query Cypher para el grafo.

- Desarrollar una función que realice llamadas a un **LLM**:
    - Justificar donde va a estar alojado el LLM (Local o en la nube).
    - Analizar el LLM a utilizar y justificar su elección.

- Integrar para la **Generación** y **Conversación**:
    - Integra todos los componentes (clasificador, recuperadores y un LLM generador) en un bucle conversacional.
    - El sistema debe soportar memoria para mantener el contexto de la conversación.
    - Debe responder en el mismo idioma de la consulta.
    - Si no encuentra información relevante, el LLM debe generar una respuesta sugiriendo al usuario reformular su pregunta.

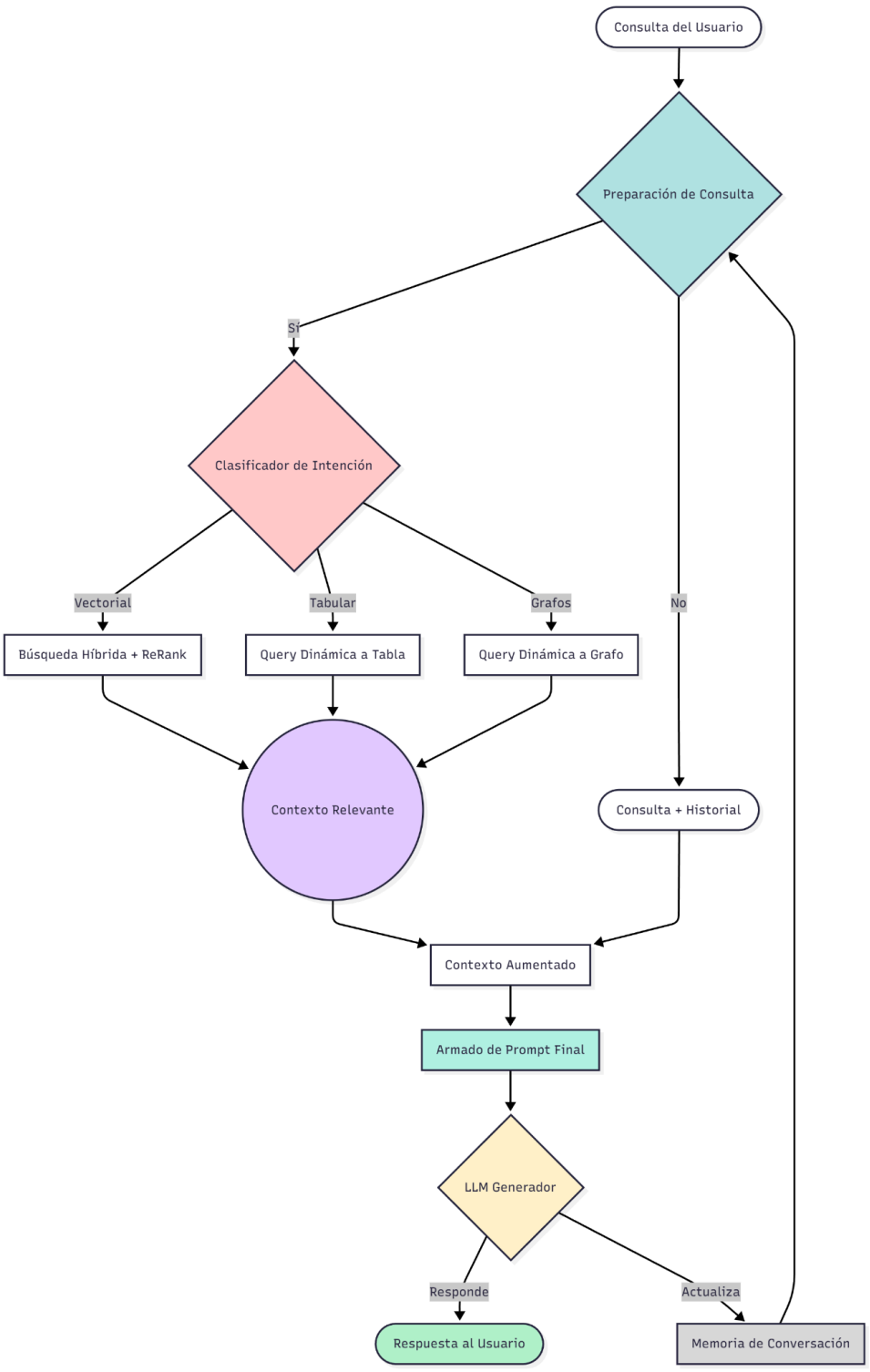


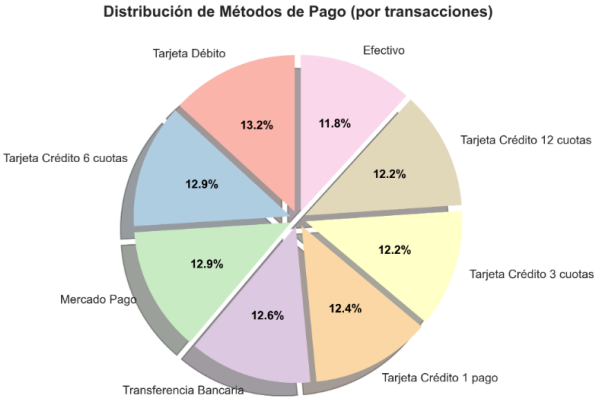
Diagrama de ejemplo del RAG

Ejercicio 2 - Evolución del RAG a un Agente Autónomo

En este ejercicio, se debe transformar el sistema RAG en un agente inteligente que pueda decidir qué herramientas utilizar para responder preguntas complejas.

Requisitos y pasos:

- Utilizar **Langchain** para implementar un agente basado en el paradigma ReAct.
- Creación de Herramientas (**Tools**):
    - Encapsula la lógica de búsqueda de cada fuente de datos del Ejercicio 1 en herramientas independientes:
        - **doc_search()**: Busca en los documentos de texto (con búsqueda híbrida y re-rank).
        - **table_search()**: Realiza consultas dinámicas a los datos tabulares.
        - **graph_search()**: Realiza consultas dinámicas a la base de datos de grafos.
- Adicionalmente, implementa una cuarta herramienta utilizando las integraciones de Langchain:
    - **analytics_tool**:
        - Para búsqueda en una base de datos SQL y generación de gráficos con matplotlib.
        - Por ejemplo:
            - El usuario consulta: "*Dame un gráfico sobre la distribución de los medios de pagos en base de las ventas realizadas*"
            - La herramienta genera un gráfico que puede enseñarlo o guardarlo en el directorio de trabajo actual:



- Construye un prompt de sistema cuidadosamente diseñado que le indique al agente el propósito de cada herramienta y cómo debe razonar (Thought) para elegir una o varias herramientas (Action) y así construir una respuesta completa.

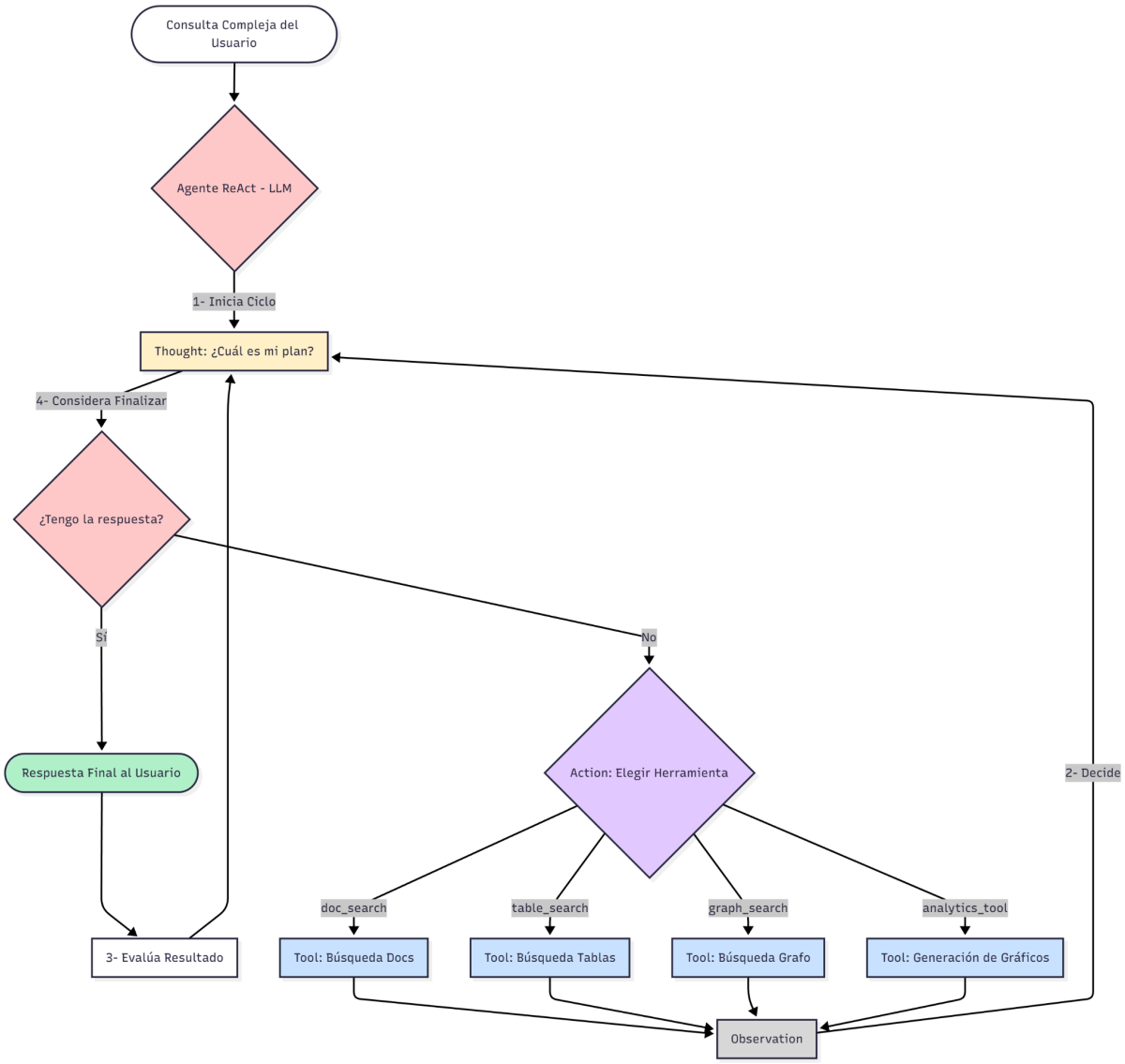


Diagrama de ejemplo del Agente ReAct

Presentación del informe

El informe debe estar en formato **PDF** y acatar los siguientes puntos:

- Debe contener justificaciones de decisión de los modelos utilizados:
  - Clasificador
  - Modelo de embedding
  - Text Splitter
  - Modelo de lenguaje
- Contener resultados de ejecución de ambos ejercicios, por lo menos 5 pruebas en cada uno en donde se evalúe **todo** el potencial del chatbot desarrollado.
- Explica qué mejoras realizarías para perfeccionar el rendimiento del agente y solucionar los fallos detectados.
- Bibliografía extra en caso de haber consultado.