HAIA: Text2SQL local
Interactuar en lenguaje natural con una base de datos

Luis Miguel Carmona Pérez
Alberto Sánchez Sanchez

Descripción

En este proyecto se va a investigar el campo de la conversión de preguntas en lenguaje natural a consultas SQL (Text2SQL o NL2SQL). El objetivo final del proyecto es desarrollar un sistema en el que se pueda consultar información de una base de datos de Fórmula 1. Adicionalmente, se va a hacer una pequeña investigación del estado del arte de modelos de Text2SQL y sus posibles aplicaciones. Nuestro sistema propone hacerlo sin el uso de APIs como las de OpenAI o Gemini, y utilizar modelos `Text2SQL` que se ejecuten en local, optimizando el uso de recursos y empleando soluciones gratuitas y de código abierto.

1. Introducción

1.1. Contexto

Una base de datos contiene mucha información estructurada, pero para acceder a ella es necesario tener conocimientos técnicos en lenguajes que permitan consultarlas, como puede ser SQL. Por otra parte, los grandes modelos de lenguaje (LLM) sufren frecuentemente "alucinaciones", es decir, es común que se inventen información cuando no conocen la respuesta o no están seguros de ella, más aún si la información a partir de la que generan la respuesta es muy extensa.

De estas dos ideas surge una solución para generar información de manera precisa, sin "alucinaciones" ni información inventada directamente por el modelo: el `Text2SQL` o `NL2SQL`. Esta tarea de los modelos de lenguajes consiste en generar una consulta para pedir la información directamente a la base de datos, en lugar de tener que generar una respuesta. En otros casos, el prompt contiene toda la información de la base de datos, lo cual incrementa la probabilidad de errores e induce al modelo a generar información errónea; en cambio, en nuestro caso, el prompt contiene el esquema de la base de datos, una serie de instrucciones y explicaciones para saber generar la consulta SQL y la

pregunta en lenguaje natural que queramos hacer. De esta forma, la información final será específicamente la que contiene la base de datos.

1.2. Objetivo del trabajo

Este trabajo tiene el objetivo final de desarrollar una pequeña aplicación donde se puedan hacer preguntas sobre Fórmula 1 y el sistema genere respuestas precisas, todo ello mediante el uso de modelos `Text2SQL` y la consulta de una base de datos diseñada específicamente para esta tarea.

1. Análisis de la literatura y el estado del arte de modelos `Text2SQL` y sus aplicaciones.
2. Creación de una base de datos con información sobre pilotos, carreras, equipos... de Fórmula 1.
3. Desarrollo del sistema principal para la consulta de información.

1.3. Estructura de trabajo

El documento se reorganiza de la siguiente manera:

- Primero se revisa el estado del arte en modelos `Text2SQL`.
- Luego se detalla la metodología de desarrollo.
- A continuación, se presentan los resultados.
- Finalmente, se discuten las conclusiones y posibles líneas futuras de mejora.

2. Estado del Arte

2.1. Modelos `Text2SQL`

Los modelos de `Text2SQL` se basan principalmente en técnicas de procesamiento de lenguaje natural y aprendizaje automático. Estos modelos están diseñados para convertir preguntas formuladas en lenguaje natural en consultas SQL que se pueden ejecutar sobre una base de datos estructurada. La conversión de texto a SQL es una tarea compleja, ya que implica no solo la comprensión del lenguaje natural, sino también el conocimiento de la estructura de la base de datos y cómo traducir esa estructura en una consulta válida.

Actualmente, los modelos `Text2SQL` utilizan redes neuronales profundas basadas en arquitecturas `seq2seq` y `Transformers`. Los `Transformers` han demostrado ser más efectivos para tareas de traducción entre lenguajes y de generación de texto debido a su capacidad para procesar información de forma paralela y capturar dependencias a largo plazo mediante mecanismos de atención. Modelos como BERT y T5 se han utilizado con éxito en la tarea de `Text2SQL`, ya que permiten obtener representaciones contextuales más ricas del texto, lo que mejora la comprensión de las consultas de lenguaje natural y la generación de la consulta SQL correspondiente.

En la mayoría de los enfoques modernos, se hace uso de modelos preentrenados como BERT, T5 o incluso GPT, que son entrenados previamente en grandes cantidades de datos no etiquetados. Luego, estos modelos se someten a un proceso de fine-tuning en un conjunto de datos específico de pares de pregunta-consulta SQL. Este enfoque ha permitido una mejora significativa en la precisión de los modelos `Text2SQL`, ya que los modelos preentrenados ya han aprendido representaciones ricas del lenguaje que pueden ser afinadas para tareas concretas.

En enfoques más recientes se integra información sobre el esquema de la base de datos durante el entrenamiento, lo que permite al modelo aprender no solo las relaciones semánticas entre las palabras, sino también las relaciones estructurales entre las tablas y las columnas de la base de datos. Esto mejora la capacidad del modelo para generar consultas SQL que se ajusten correctamente a las bases de datos específicas con las que se está trabajando.

2.2. Benchmarks

Para la evaluación de los modelos se emplean benchmarks específicos diseñados para comprobar si la generación de las consultas SQL generadas o las respuestas finales de la base de datos son precisas. Los más importantes en la actualidad son:

- Spider2.0 [2]: Un conjunto de datos que ha sido ampliamente utilizado para entrenar y evaluar modelos `Text2SQL`. Está compuesto por preguntas en lenguaje natural junto con consultas SQL correspondientes para una amplia gama de bases de datos.
- BIRD [3]: Un benchmark reciente centrado en consultas complejas en múltiples tablas, diseñado para evaluar modelos en escenarios realistas con estructuras de bases de datos ricas y variadas.

2.3. Soluciones y modelos actuales

En los últimos años han surgido diferentes alternativas para la tarea de `Text2SQL` que han evolucionado desde soluciones más simples basadas en modelos secuencia a secuencia, hasta arquitecturas más sofisticadas. Algunos de los primeros modelos en

abordar este problema fueron `Seq2SQL` y `SQLNet` que abordaban el problema utilizando redes neuronales con mecanismos de atención, pero presentaban ciertas limitaciones cuando se enfrentaban a esquemas complejos o a consultas que requerían múltiples operaciones.

Más adelante, se introdujeron modelos como `RAT-SQL` que incorporaban el esquema de la base de datos durante la generación, mejorando notablemente el modelado de las relaciones entre tablas, columnas y condiciones, lo cual se veía reflejado en las precisiones en benchmarks como Spider.

En la actualidad, podemos encontrar desde software privado de grandes compañías como Google o Meta, hasta modelos de código abierto diseñados por equipos de investigación de diferentes universidades. Una de las soluciones más populares son los grandes modelos de lenguaje de estas compañías, los cuales, a pesar de no estar entrenados específicamente para esta tarea, obtienen resultados cercanos a las precisiones de modelos `Text2SQL`. Estos LLMs son accesibles desde APIs a muy bajo coste, como es el caso de Gemini o GPT4, y cuentan con la potencia de cómputo y la cantidad de parámetros que ofrecen estas empresas, de manera que es imposible competir con recursos locales.

En este proyecto se plantean los objetivos con la restricción de ser ejecutados en local, por este motivo se ha optado por el modelo **XiYanSQL-Coder**. Este modelo es una versión de **QwenCoder** con un *fine-tuning* específico para esta tarea; además, es el estado del arte en benchmarks como Spider y permite ser ejecutado localmente.

2.4. Aplicaciones Relevantes

El uso de sistemas `Text2SQL` tiene un amplio rango de aplicaciones en sectores donde el acceso a información estructurada es clave y donde no todos los usuarios cuentan con los conocimientos técnicos necesarios para realizar consultas SQL. Una de las aplicaciones más directas es en sistemas de soporte a la toma de decisiones empresariales, donde empleados de departamentos como marketing o ventas pueden consultar bases de datos sin depender de equipos técnicos.

Otra aplicación prometedora es en análisis de datos. Por ejemplo, un ejemplo real sería en una fábrica, donde toda la información de la planta (maquinaria, alarmas, registros...) se almacena en una base de datos. Los técnicos de mantenimiento y demás trabajadores sin conocimientos específicos sobre bases de datos podrían hacer consultas directamente sobre el sistema, y la información devuelta sería precisa, ya que viene directamente de la base de datos.

3. Metodología

3.1. Enfoque del Trabajo

El desarrollo del proyecto se divide en dos partes diferenciadas: en primer lugar, la construcción de la base de datos; y en segundo lugar, la creación del sistema que incorporará, desde la interfaz donde realizar las preguntas, hasta la consulta en la base de datos creada.

3.2. Tecnologías y Herramientas

Se ha decidido utilizar una base de datos SQLite creada mediante la herramienta de Python `sqlite3` y la API `Ergast-py`, en la que se han hecho varias llamadas a `Ergast-py` para obtener toda la información disponible, para así guardarla en una base de datos SQLite.

El modelo `Text2SQL` utilizado es **XiYanSQL-Coder** [1] con 3B de parámetros. Este modelo nos permite ejecutarlo localmente, sin necesidad de utilizar APIs comerciales y cuenta con una precisión que actualmente es el estado del arte en modelos para esta tarea.

Para la implementación del sistema, se ha optado por una implementación propia y se descarta utilizar librerías como *langchain*, ya que preferimos tener mayor control en la implementación. Para la creación de la base de datos se ha utilizado SQLite y para su manejo y consulta se utiliza *sqlalchemy*. El modelo de XiYan se descarga y ejecuta mediante la librería de *transformers* que nos proporciona HuggingFace. Por último, se emplea la librería *streamlit* para crear una interfaz web cómoda y accesible de forma sencilla.

4. Desarrollo

4.1. Creación de la base de datos

La base de datos, tal y como se ha explicado en el punto anterior, se ha creado utilizando SQLite como el sistema de creación y gestión de la misma. Se ha utilizado esta tecnología por su simplicidad, portabilidad y facilidad de implementación con el entorno ideado para este proyecto. También ofrece tablas relacionales en sus bases de datos, característica esencial en este proyecto.

La base de datos que se utilizará en este proyecto contiene todos los resultados de todas las carreras que se han disputado en el *FIA Formula One World Championship* (Fórmula 1) a lo largo de su historia. El año de inicio es el mismo de la competición:

1950. El último año disponible en la API es 2022, por lo cual, ese será también el último año disponible en nuestra base de datos. Aparte de los resultados, la base de datos también contiene información inherente a estos resultados (información de los pilotos que participan en la carrera, de las escuderías, de los circuitos y países donde se albergan las carreras, etc.). Para la creación de la base de datos, se ha diseñado con las siguientes tablas:

Cuadro 1: Tabla: Piloto

Campo	**Tipo**	**Descripción**
id	TEXT (PK)	Identificador único del piloto
nombre	TEXT	Nombre del piloto
apellido	TEXT	Apellido del piloto
fecha_nacimiento	TEXT	Fecha de nacimiento
nacionalidad	TEXT	Nacionalidad

La tabla Piloto contiene toda la información relevante sobre el piloto que disputa un gran premio, tanto sus nombres y apellidos como su fecha de nacimiento y nacionalidad. Datos relevantes como número de victorias, *poles* o similares no se indican debido a que pueden calcularse directamente con la base de datos.

Cuadro 2: Tabla: Escuderia

Campo	**Tipo**	**Descripción**
id	TEXT (PK)	Identificador único de la escudería
nombre	TEXT	Nombre de la escudería
nacionalidad	TEXT	Nacionalidad de la escudería

La tabla Escudería contiene toda la información relevante sobre la escudería que disputa la carrera. De nuevo, hay datos relevantes que no se incluyen porque se pueden calcular directamente mediante una consulta SQL.

Cuadro 3: Tabla: PilotoEscuderia

Campo	**Tipo**	**Descripción**
piloto_id	TEXT (PK, FK)	ID del piloto (relación con Piloto)
escuderia_id	TEXT (PK, FK)	ID de la escudería (relación con Escuderia)

La tabla PilotoEscuderia es una tabla relacional que sirve para establecer la relación entre un piloto y las escuderías para las que ha competido a lo largo de su carrera deportiva, así la relación de una escudería con todos los pilotos que han competido bajo su nombre a lo largo de su historia.

Cuadro 4: Tabla: GP

Campo	**Tipo**	**Descripción**
id	TEXT (PK)	Identificador del Gran Premio
año	INTEGER	Año del evento
dia	INTEGER	Día del evento
mes	INTEGER	Mes del evento
hora	TEXT	Hora de inicio
circuito	TEXT	Nombre del circuito
granpremio	TEXT	Nombre del GP (por país o ciudad)

La tabla GP contiene toda la información necesaria sobre un Gran Premio de Fórmula 1. Un Gran Premio de Fórmula 1 es el conjunto de eventos que ocurren en un fin de semana. En este caso, solo se muestra la información de la carrera, tales como fecha y hora.

Cuadro 5: Tabla: ResultadoGP

Campo	**Tipo**	**Descripción**
gp_id	TEXT (PK, FK)	ID del GP
piloto_id	TEXT (PK, FK)	ID del piloto
escuderia_id	TEXT (FK)	ID de la escudería
posicion	INTEGER	Posición final
parrilla	INTEGER	Posición de salida
tiempo	TEXT	Tiempo final (o diferencia)
puntos	FLOAT	Puntos obtenidos

La tabla ResultadoGP es una tabla relacional que enlaza a un piloto con una carrera, indicando su resultado, y datos diferentes inherentes a la carrera del piloto, como puntos obtenidos o la diferencia o *gap* con el piloto vencedor.

Cuadro 6: Tabla: Circuito

Campo	**Tipo**	**Descripción**
id	TEXT (PK)	Identificador del circuito
nombre	TEXT	Nombre del circuito
ciudad	TEXT	Ciudad del circuito
pais	TEXT	País del circuito

Por último, la tabla Circuito contiene información relevante sobre el circuito donde se disputa la carrera.

Una vez diseñada la base de datos, se dispuso a crearla mediante la API de Python `sqlite3`. A continuación, se realizó una búsqueda en la API de `Ergast-py` por cada carrera que hay en cada año en un rango entre 1950 y 2022. Una vez obtenida la respuesta de la API, se decidió seleccionar la información a introducir en cada una de las tablas, se introdujo en las tablas y se cerró la base de datos.

4.2. Descarga del modelo Text2SQL

Los modelos XiYanSQL-coder están accesibles desde diferentes servicios; en nuestro caso, se ha decidido utilizar la librería *transformers* de HuggingFace para cargar el modelo y la librería *PyTorch* para hacer la inferencia. Se ha escogido el modelo de 3B de parámetros para poder ejecutarlo de forma local; este modelo devuelve buenos resultados en los benchmarks públicos y es posible ejecutarlo localmente con una tarjeta gráfica dedicada.

4.3. Estructura del desarrollo

El desarrollo se ha decidido dividir por servicios: por una parte, el servicio que se conecta con la base de datos y realiza las consultas SQL; por otra parte, el servicio unido a la generación de la respuesta, que se encarga de recibir la pregunta, tokenizarla y generar una consulta SQL válida; y por último, la interfaz gráfica desde la que se pueden realizar las preguntas de forma cómoda y visualizar las respuestas.

El código de la aplicación está disponible en el repositorio GitHub `https://github.com/luismi2801/2425_HAIA.git`.

4.4. Interfaz gráfica

Para poder crear una interfaz sencilla y visual para interactuar con el sistema, se ha decidido utilizar la librería *streamlit* que nos ofrece herramientas para crear interfaces web desde Python. La interfaz consistirá en un cuadro de texto para introducir el input, es decir, la pregunta que queremos realizar. Una vez introducida una pregunta, se escribirá en el registro la pregunta realizada, la SQL generada y la respuesta, en formato de tabla.

Figura 1: Interfaz gráfica de la aplicación

4.5. Desarrollo de un corpus de evaluación

Para evaluar el sistema, se va a crear un dataset con pares pregunta-respuesta que nos permita calcular métricas sobre la precisión de nuestro modelo. También podremos medir otros modelos y APIs para comparar resultados.

5. Resultados y Evaluación

5.1. Precisión del Modelo

Para el cálculo de la precisión del modelo, debido a la falta de un *dataset* especializado en *queries* de Fórmula 1, se ha decidido proceder con la siguiente metodología: se han seleccionado un total de 50 preguntas en lenguaje natural sobre la base de datos del proyecto. Se contarán como aciertos las veces que el modelo devuelva la respuesta correcta, y como fallos cuando devuelva una respuesta errónea o no devuelva una respuesta.

Tras realizar la evaluación, se obtuvo una precisión de un **76 %**. Hay que tener en cuenta que se trata de un modelo sin *fine-tuning*, por lo que obtener esa precisión es un resultado bastante prometedor.

5.2. Tiempo de Respuesta y Recursos

Los tiempos de respuesta desde la API son de escasos segundos en un equipo con una tarjeta gráfica dedicada. Si se realiza solamente mediante CPU, el proceso de inferencia puede tardar de entre 10 a 30 segundos, dependiendo de la complejidad de la pregunta realizada.

5.3. Ejemplos de Consultas

- ¿En qué circuito se corre el GP de Gran Bretaña?
- ¿Qué piloto ha terminado más veces en el segundo lugar?
- ¿Quién terminó quinto en el GP de Hungría 2003?

6. Conclusiones

6.1. Resumen

Este proyecto se centra en el desarrollo de un sistema que permita la interacción en lenguaje natural del usuario con una base de datos estructurada. El enfoque se basa en `Text2SQL`, cuyo objetivo es traducir las preguntas en lenguaje natural a consultas SQL válidas que pueden ejecutarse sobre una base de datos.

Para alcanzar este objetivo, se ha realizado una revisión exhaustiva del estado del arte en modelos de Text2SQL. Se han analizado *benchmarks* relevantes como Spider2.0 o BIRD, y se ha seleccionado el modelo `XiYanSQL-Coder` para ser implementado en el proyecto.

El trabajo se divide en dos etapas: la construcción de una base de datos con información extraída de `Ergast-py`, y el desarrollo del sistema de consulta. La base de datos se implementó en SQLite. En cuanto al sistema de procesamiento, se optó por una implementación propia en Python para tener control total sobre cada componente.

6.2. Conclusión Final

El desarrollo de este proyecto ha permitido comprobar que es posible implementar un sistema funcional de Text2SQL en local, sin necesidad de depender de soluciones comerciales o en la nube.

La elección de `XiYanSQL-Coder` ha resultado ser adecuada, ya que combina la potencia de los transformers con un ajuste fino orientado específicamente a la tarea de

conversión Text2SQL. Su compatibilidad con la ejecución en local y su precisión en las pruebas realizadas demuestran que, sin haberse realizado ningún tipo de *fine-tuning*, el modelo ha podido obtener puntuaciones altas.

Este proyecto también ha facilitado el acceso a datos relevantes del mundo de la Fórmula 1, como pilotos, equipos, circuitos y resultados de carreras. Este es solo uno de los múltiples ejemplos de bases de datos a las que se puede aplicar esta tecnología, haciendo mucho más accesible la información a los usuarios sin ningún tipo de experiencia en SQL.

En conclusión, este trabajo ha demostrado que es posible combinar procesamiento del lenguaje natural, diseño de bases de datos y ejecución eficiente en local para ofrecer una solución accesible, fiable y útil en contextos reales donde el acceso a la información estructurada debe ser democrático y directo, sin intermediarios técnicos ni barreras de entrada.

6.3. Trabajos Futuros

Existen varias líneas de mejora y ampliación que podrían explorarse en un futuro.

En primer lugar, una posible extensión sería la integración de un modelo de lenguaje adicional, como LLaMA 3.2 u otro LLM ligero, que actúe como explicador de consultas o resultados. Este LLM podría recibir la pregunta, la consulta SQL y los resultados de la BBDD, y generar una explicación en lenguaje natural que permitiese entender más fácilmente los resultados, ya que nuestro sistema los presenta en forma de tabla.

Otra posible mejora sería poder hacer consultas sobre información anterior. Es decir, mantener un historial con las preguntas y respuestas, y poder consultar sobre este contexto, de forma que si pregunto "¿Quién ganó el mundial de Fórmula 1 en 2018?", podría preguntar directamente "¿y en 2019?".

Finalmente sería interesante incluir validación SQL antes de su ejecución. Para prevenir errores en las consultas, o para corregirlos, sería posible añadir mecanismos que se encargaran de encontrar o corregir fallos. Esto podría hacerse mediante LLMs, pasándole la SQL junto con el fallo.

Referencias

[1] Yingqi Gao et al. «A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL». En: *arXiv preprint arXiv:2411.08599* (2024). URL: https://arxiv.org/abs/2411.08599.

[2] Fangyu Lei et al. *Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows*. 2024. arXiv: 2411.07763 [cs.CL]. URL: https://arxiv.org/abs/2411.07763.

[3] Chien-Sheng Li et al. «BIRD: A Benchmark for Irregular and Diverse Text-to-SQL Evaluation». En: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*. 2023. URL: https://arxiv.org/abs/2306.01627.