🚗 Vehicle DB AI Demo

✨ Funkcje

- Upload obrazu i klasyfikacja pojazdu (YOLO lub własny klasyfikator).
- Zadawanie pytań w języku naturalnym → automatyczne generowanie zapytań SQL SELECT.
- Automatyczna walidacja SQL (blokowanie INSERT/UPDATE/DELETE/...).
- Automatyczne wykrywanie schematu bazy SQLite i użycie go w promptach.
- Eksport wyników zapytań do CSV.

🛠️ Technologie

- Streamlit – interfejs webowy
- Ollama – lokalne LLM do generowania SQL
- SQLite – baza danych
- Pandas – obsługa tabel danych
- Pillow – obsługa obrazów
- Ultralytics YOLO –doklasyfikacji pojazdów

⚙️ Instalacja

1. Pobrać wszystkie pliki projektu np. z GitHub.
2. Pobrać model llama-3-sqlcoder-8b.Q5_K_M.gguf z linka:
   https://huggingface.co/QuantFactory/llama-3-sqlcoder-8b-GGUF/blob/main/llama-3-sqlcoder-8b.Q5_K_M.gguf - opis modelu
   https://huggingface.co/QuantFactory/llama-3-sqlcoder-8b-GGUF/resolve/main/llama-3-sqlcoder-8b.Q5_K_M.gguf?download=true – link do pobrania
3. Pobrać model YOLO do klasyfikacji:
   https://huggingface.co/Ultralytics/YOLO11/blob/5e0da476eb5def45e8080bd4b92ea63f0b16974c/yolo11m.pt - opis modelu
   https://huggingface.co/Ultralytics/YOLO11/resolve/5e0da476eb5def45e8080bd4b92ea63f0b16974c/yolo11m.pt?download=true – link do pobrania
4. Przenieść pobrane modele do folderu projektu (tam gdzie jest plik system.py).
5. Pobrać i zainstalować Ollama z linka:
   https://ollama.com/download/windows
6. Okienko z Ollama, które się otworzy można wyłączyć.
7. Otworzyć konsolę poleceń CMD.

8. Przejść do ścieżki, w której znajduje się projekt (plik system.py).
9. Zainstalować wymagane pakiety poleceniem: *pip install streamlit ollama pandas pillow ultralytics* LUB *pip install -r requirements.txt*
10. Wpisać w CMD polecenie: *ollama create sqlcoder -f Modelfile*
11. W CMD wpisać polecenie uruchamiające projekt: *streamlit run web_app_streamlit.py*
12. ALTERNATYWNIE wszystkie powyższe komendy CMD można uruchomić w terminalu np. PyCharm.
13. W przypadku błędu „Error: model requires more system memory" ze strony https://huggingface.co/QuantFactory/llama-3-sqlcoder-8b-GGUF/tree/main można pobrać inny lżejszy model o większej kwantyzacji np. Q4_K_M albo Q3_K_M – działają one szybciej i wymagają mniej pamięci ale mają mniejszą jakość odpowiedzi. W przypadku pobrania innego modelu postępować należy analogicznie co powyżej z drobnymi zmianami:
    - Usunąć poprzedni model poleceniem: *ollama rm sqlcoder*
    - Zmienić w pliku Modelfile.txt nazwę na tą pobranego modelu

---

📂 **Struktura**

- system.py – główny skrypt aplikacji
- database.db – przykładowa baza SQLite
- images/ – katalog na obrazy pojazdów
- model klasyfikujący
- model NL → SQL

🔧 **Konfiguracja**

- Model LLM: llama-3 fine-tuningowany do obsługi zapytań NL->SQL.
- Klasyfikator pojazdów: yolo11m.pt
- Oba modele można łatwo podmienić na inne.

⚠️ **Uwagi**

- Dozwolone są tylko SELECT w SQL (dla bezpieczeństwa).
- Agent zawsze dodaje vehicle_id do wyników zapytań związanych z pojazdami.
- Schemat bazy jest automatycznie ekstraktowany i skracany, by zmniejszyć prompt.

📥 **Eksport wyników**

- Wyniki zapytań SQL można pobrać jako plik **CSV**.

Kilka słów o tworzeniu aplikacji:

1. **Wybór zbioru danych i typu modelu**
   Na początku rozważałem trenowanie własnego modelu klasyfikującego pojazdy. Z uwagi na brak dostępu do w pełni kompatybilnej karty graficznej i konieczność długiego czasu uczenia, wstępnie zdecydowałem się na zbiór danych **CIFAR-100**, który charakteryzuje się bardzo małymi obrazami. Rozwiązanie to mogłoby jednak nie sprawdzić się przy większych obrazach i niosło spore ryzyko. Ponieważ w instrukcji dopuszczono możliwość użycia gotowych modeli, ostatecznie wybrałem gotowy model **YOLO11m.pt** – oferujący rozsądny kompromis pomiędzy szybkością i dokładnością.

2. **Model NL → SQL – lokalny vs API**
   W przypadku zamiany języka naturalnego na SQL rozważałem dwa podejścia: użycie płatnego modelu API (np. OpenAI) albo modelu lokalnego. Każde rozwiązanie miało swoje zalety i wady. Ostatecznie zdecydowałem się na model lokalny – zapewniał on wystarczająco dobre wyniki, a jednocześnie nie wiązał się z kosztami i był bezpieczniejszy przy pracy z danymi wrażliwymi. Minusem była nieco niższa szybkość działania względem API, konieczność przetestowania paru modeli by wybrać możliwie najlepszy oraz napisanie promptów najlepiej dostosowanych do wybranego modelu.

3. **Architektura projektu i interfejs**
   Przygotowałem dwa odrębne programy: jeden do analizy obrazów, a drugi do tłumaczenia zapytań z języka naturalnego na SQL. Do stworzenia interfejsu webowego posłużyłem się narzędziem AI, a następnie samodzielnie poprawiłem wygenerowany kod i połączyłem go z przygotowanymi modułami. Następnie wprowadziłem liczne usprawnienia, takie jak walidacja zapytań SQL czy dopracowane prompty/instrukcje dla modelu.

4. **Walidacja i bezpieczeństwo SQL**
   Aby uzyskać jak najlepsze i bezpieczne wyniki, wygenerowany kod SQL jest sprawdzany względem schematu bazy danych. Jeśli model użyje nieistniejących tabel lub kolumn, albo zapytanie zwróciłby błąd podczas uruchomienia, kod SQL jest ponownie generowany (maksymalnie 3 próby, z różnymi wartościami seed i temperatury). Dodatkowo, zapytania zawierające niedozwolone instrukcje (np. DROP, INSERT, UPDATE) są odrzucane i nie są wykonywane.

5. **Komentarze w kodzie**
   Ponieważ projekt miał charakter testowy, aby przyspieszyć przygotowanie i ułatwić sprawdzanie kodu, większość komentarzy została wygenerowana przez ChatGPT. Dotyczy to również wstępnego szkicu tego pliku opisującego projekt (stąd np. ikonki w niektórych miejscach 😊).

6. **Elastyczność projektu**
   Program umożliwia łatwą zmianę wykorzystywanych modeli AI oraz bazy danych za pomocą panelu z lewej strony. W przypadku całkowitej zmiany schematu bazy lub modelu AI konieczne byłoby przygotowanie nowych promptów/instrukcji, aby nadal osiągać optymalne wyniki, jednak sama struktura aplikacji pozostaje uniwersalna.

7. **Dalsze szczegóły**
   Chętnie opowiem o innych drobniejszych problemach napotkanych w trakcie pracy nad projektem oraz o zastosowanych rozwiązaniach podczas rozmowy rekrutacyjnej. Większość z nich dotyczy nieprzestrzegania przez model zasad przedstawionych w promptach instrukcji lub instrukcji zrozumiałych dla człowieka a jednak nie dla modelu AI.