Politechnika Łódzka
Wydział Elektrotechniki Elektroniki
Informatyki i Automatyki

inż. Łukasz Klimkiewicz

244822

PRACA DYPLOMOWA

magisterska

na kierunku Sztuczna Inteligencja i Uczenie Maszynowe

Tłumaczenie zapytań w języku polskim na SQL

Instytut Informatyki Stosowanej

Promotor: prof. dr hab. Szymon Grabowski

Łódź 2024

Streszczenie

Niniejsza praca dyplomowa dotyczy istotnego obecnie problemu tłumaczenia zapytań z języka naturalnego na SQL. Wyróżnia się wykorzystaniem polskiego w roli języka naturalnego, co do tej pory nie zyskało zainteresowania. W jej ramach stworzono odpowiednie zbiory danych, przeprowadzono eksperymenty i opracowano użyteczną aplikację.

Pracę rozpoczęto od przeanalizowania istniejących zbiorów danych ze szczególnym zwróceniem uwagi na zbiór `Spider` i jego tłumaczenia. W celu stworzenia polskich odpowiedników obrano podejście polegające na tłumaczeniu maszynowym oraz skryptowym generowaniu różnych wariantów. W dalszej kolejności przeprowadzono przegląd problemu `Text-to-SQL` i wybrano 4 modele, na których przeprowadzono dalsze eksperymenty. Ujawniły one wysoką skuteczność rozwiązań `RESDSQL` oraz `C3`, które następnie zostały zintegrowane z wygodną aplikacją, umożliwiającą generowanie i wykonywanie zapytań. Jej użytkownicy, pomimo zauważenia pewnych problemów, docenili praktyczność opracowanego rozwiązania.

Słowa kluczowe: Text-to-SQL, SQL, tłumaczenie, język polski, Spider

Abstract

This thesis addresses the currently relevant problem of translating queries from natural language to SQL. It is distinguished by the use of Polish in the role of a natural language, which has not received attention so far. It involved the creation of relevant datasets, experimentation and development of a useful application.

The work began by analyzing existing datasets with particular attention to the `Spider` dataset and its translations. The approach of machine translation and scripted generation of various variants was taken to create Polish equivalents. The `Text-to-SQL` problem was then reviewed and 4 models were selected on which further experiments were conducted. They revealed the high accuracy of `RESDSQL` and `C3` solutions, which were then integrated into a convenient application for generating and executing queries. Its users, despite noticing some problems, appreciated the practicality of the developed system.

Keywords: Text-to-SQL, SQL, translation, Polish language, Spider

Spis treści

1 Wprowadzenie

W dzisiejszych czasach dane odgrywają tak ważną rolę jak nigdy wcześniej. Niemal wszystkie istotne działania podejmowane przez ludzi na nich się opierają, a ich ilość oraz stopień skomplikowania rośnie z dnia na dzień. Duża część informacji przechowywania jest w relacyjnych bazach danych, lecz dostęp do nich stanowi istotny problem. Większość istniejących narzędzi stworzona jest bowiem albo dla amatorów i bardzo ograniczona, albo dla ekspertów i niezwykle skomplikowana. Rozwiązaniem tego dylematu wydają się interfejsy do baz danych w postaci języka naturalnego (ang. Natural Language Interfaces for Databases, NLIDBs), które pozwalają je odpytywać w naturalny dla ludzi sposób. Istniejące prace w zdecydowanej większości utożsamiają jednak język naturalny z językiem angielskim, co powoduje wykluczenia dużej grupy użytkowników. To właśnie powody, dla których w niniejszej pracy postanowiono zająć się problemem tłumaczenia zapytań z języka polskiego na SQL.

1.1 Wstęp

Automatyczne generowanie zapytań SQL na podstawie naturalnych żądań to zagadnienie, którego istotność została już dawno temu dostrzeżona. Zgodnie z przeglądem dokonanym w artykule *Natural Language Interfaces to Data* (Quamar i in. 2022) pierwsze tego typu prymitywne rozwiązania powstawały już na początku lat dziewięćdziesiątych. Bazowały na ręcznie tworzonych regułach i konstruowały zapytania na podstawie wychwyconych słów kluczowych, czy też znalezionych podczas parsowania zdania naturalnego zależności. Niemożliwym okazywało się jednak zdefiniowanie reguł, które pokryłyby wszystkie możliwe żądania użytkowników.

Wielkim przełomem dla problemu generowania zapytań SQL okazało się zaprzęgnięcie do tego celu uczenia maszynowego. Podejście takie nie wymaga manualnego definiowania reguł, lecz dostępu do dużej ilości przykładów, co stanowiło do pewnego momentu problem. Został on rozwiązany wraz z publikacją dużych zbiorów danych, do których należy WikiSQL (Zhong i in. 2017a) oraz bardziej rozbudowany Spider (T. Yu, Zhang, K. Yang i in. 2018a). W szczególności na bazie tego ostatniego powstało wiele modeli uczenia maszynowego. To właśnie takie podejście, polegające na uczeniu na podstawie danych, pozwala dzisiaj osiągać najlepsze rezultaty.

Spider został stworzony jako zbiór anglojęzyczny, co wydaje się dobrym i oczywistym wyborem ze względu na popularność i powszechność tego języka. Z wykorzystaniem takich danych możliwe jest jednak budowanie rozwiązań rozumiejących teksty wyłącznie angielskie, co stanowi

utrudnienie dla osób nieznających tego języka w dostatecznym stopniu. Przyczyniło się to do podjęcia pracy nad przetłumaczeniem zbioru Spider na inne języki, w wyniku czego dostępne jest dzisiaj publicznie tłumaczenie chińskie (Min i in. 2019), wietnamskie (A. T. Nguyen i in. 2020), portugalskie (José i in. 2021), rosyjskie (Bakshandaeva i in. 2022) oraz zbiór MultiSpider (Dou i in. 2023), zawierający dodatkowo wersję niemiecką, francuską, hiszpańską oraz japońską. Generowanie zapytań SQL dla języka polskiego nie zyskało jednak zainteresowania, aż do chwili obecnej.

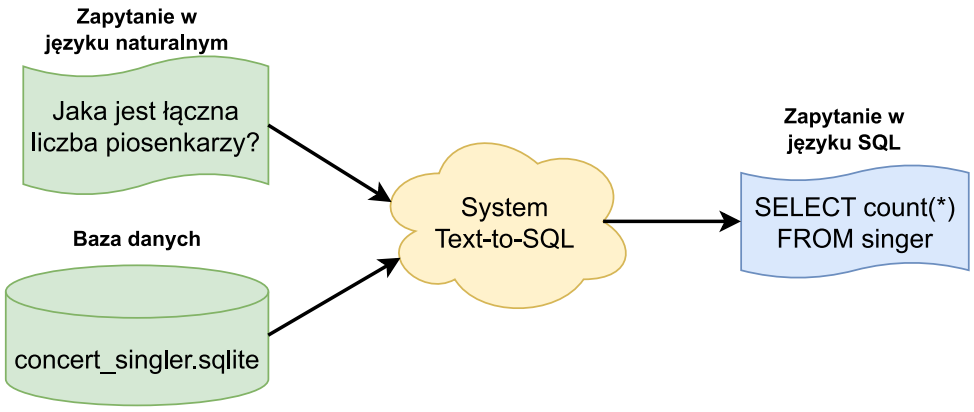
Przewidywanie zapytania SQL na podstawie pojedynczego zapytania naturalnego nie stanowi mimo wszystko ostatecznego celu w kwestii ułatwienia dostępu do danych. Dalszym rozwinięciem tego pomysłu są powstające już systemy konwersacyjne, które pozwalają na bardziej naturalną komunikację. Podczas generowania zapytań SQL uwzględniają one kontekst tworzony przez wcześniejsze wiadomości, w razie niejasności mogą zadawać pomocnicze pytania, czy też informować o trudnościach ze skonstruowaniem żądanej instrukcji SQL. Takie rozwiązania również wykorzystują uczenie maszynowe, a do ich rozwoju przyczyniło się powstanie zbiorów SParC (T. Yu, Zhang, Yasunaga i in. 2019) oraz CoSQL (T. Yu, Zhang, Er i in. 2019). Innym problemem jest poprawna interpretacja przez użytkowników zwracanych w wyniku wygenerowania i wykonania zapytań danych tabelarycznych. Poczynione zostały więc również wysiłki w kierunku stworzenia systemów, które poza generowaniem zapytań SQL mogłyby je wykonać i odczytane informacje prezentować użytkownikom w postaci zrozumiałej wiadomości tekstowej (T. Yu, Zhang, Er i in. 2019).

1.2 Opis problemu

Tłumaczenie zapytań z języka naturalnego na SQL znane jest obecnie w literaturze pod nazwami Text-to-SQL, NL-to-SQL, czy też Text2SQL, NL2SQL. Pierwsze określenie wydaje się najpopularniejsze i będzie wykorzystywane w dalszej części pracy. Problem ten stanowi uszczegółowienie bardziej ogólnego zagadnienia nazywanego parsowaniem semantycznym, polegającym na tłumaczenia tekstu zapisanego językiem naturalnym do dowolnej postaci formalnej.

Zagadnienie Text-to-SQL można zdefiniować następująco: *Mając podane zapytanie w języku naturalnym oraz posiadając dostęp do pewnej relacyjnej bazy danych, zwróć zapytanie SQL, które będzie dla tej bazy poprawne i równoważne zapytaniu naturalnemu*. Tłumaczenie zapytań

SQL opisywane w dalszej części pracy oraz zawarte w jej tytule rozumiane jest zgodnie z tą definicją. Prosty diagram stanowiący dla niej ilustrację przedstawiono na rysunku 1.1.



Rys. 1.1: Problem `Text-to-SQL`

Zapytanie naturalne może przyjmować wiele różnych postaci. Oczywistą formą są zdania rozkazujące, ponieważ bardzo dobrze mapują się na zapytania SQL, które mają strukturę zbliżoną właśnie do zdań rozkazujących. Równie dobrze rolę zapytań naturalnych mogą pełnić jednak zdania pytające, w których użytkownik zapytuje system o potrzebną informację. Dopuszczalne jest również wykorzystanie w tej roli bezokoliczników czy też pojedynczych słów kluczowych.

1.3 Cel i zakres pracy

Celem niniejszej pracy dyplomowej jest wykorzystanie technik uczenia maszynowego, aby opracować rozwiązanie dokonujące tłumaczenia zapytań z języka polskiego na SQL oraz jego przetestowanie. System ten ma rozwiązywać zdefiniowany we wcześniejszej sekcji problem. W szczególności ma działać w sposób bezkontekstowy, nieumożliwiający interakcji w konwersacyjnej formie, co stanowi problem trudniejszy. Istotnym elementem testów ma być ocena użyteczności powstałego narzędzia dokonana przez potencjalnych użytkowników.

Pośrednim celem pracy, niezbędnym jednak do stworzenia pożądanego rozwiązania, jest opracowanie polskich zbiorów danych. Ma to zostać dokonane poprzez tłumaczenie istniejących zbiorów anglojęzycznych, w szczególności zbioru `Spider`, ponieważ na nim większość tego typu modeli jest uczona. W zakres pracy wchodzi przeprowadzenie eksperymentów w celu lepszego zrozumienia działania modeli `Text-to-SQL`, wybranie najbardziej obiecującego, zintegrowanie go z interfejsem graficznym i finalnie przetestowanie użyteczności tak stworzonej aplikacji.

2 Wykorzystane narzędzia

Podczas realizacji tematu pracy wykorzystanych zostało wiele narzędzi rozumianych jako języki programowania, biblioteki, środowiska programistyczne, czy inne technologie. W niniejszym rozdziale przedstawione i dokładniej opisane zostały te, które odegrały kluczową rolę.

2.1 Język Python

Python[1] jest jednym z najpopularniejszych języków wykorzystywanych do analizy danych oraz uczenia maszynowego. Jest zresztą ogólnie jednym z najpopularniejszych języków, co zostało potwierdzone przez badanie *2023 Developer Survey* (Stack Overflow 2023), gdzie uplasował się na 4. miejscu w rankingu popularności.

Python wykorzystano ze względu na ogromną liczbę bibliotek, które są dla niego dostępne. Kilka spośród nich, które odegrały istotną rolę podczas realizacji niniejszego tematu, zostało opisanych w poniższych sekcjach. Ponadto umożliwia on bardzo szybkie prototypowanie i eksperymentowanie, co również zostało docenione. Jest to bowiem język wysokopoziomowy i zwalnia programistów z zajmowania się wieloma kwestiami. Jest elastyczny ze względu na swoje wieloparadygmatowe podejście: obiektowe, proceduralne, jak i funkcyjne. Ostatecznie jest językiem skryptowym i brak narzutu czasowego na kompilację znacznie skraca pętlę zwrotną między pisaniem kodu a obserwowaniem jego skutków.

2.1.1 Biblioteka PyTorch

Biblioteka `PyTorch`[2] stanowi potężne narzędzie w obszarze uczenia maszynowego i głębokiego uczenia. Pozwala na tworzenie i szkolenie różnorodnych modeli sieci neuronowych, a charakteryzuje się przy tym elastycznością i wydajnością. Jej moduły obejmują wsparcie dla operacji tensorowych, automatyczne różniczkowanie (co ułatwia proces uczenia się sieci), a także obsługę zarówno `CPU`, jak i `GPU` do przyspieszenia obliczeń. Jednym z największych atutów biblioteki `PyTorch` jest prostota w tworzeniu modeli. Dzięki intuicyjnej składni oraz dynamicznemu grafowi obliczeniowemu użytkownicy mogą szybko prototypować, testować i dostosowywać swoje modele, co sprawia, że jest szczególnie popularnym wyborem dla zastosowań badawczych.

[1] `https://python.org`
[2] `https://pytorch.org`

2.1.2 Biblioteki SpaCy i Stanza

Biblioteki SpaCy[3] i Stanza[4] to popularne dla języka Python drogi do przeprowadzenia analizy tekstów w języku naturalnym. Zostały przedstawione razem, ponieważ różnice między nimi są niewielkie i za pomocą obu można realizować niemal identyczne cele. O wykorzystaniu pierwszej lub drugiej zadecydowały mało istotne szczegóły, takie jak interfejs, który w danym momencie wydawał się wygodniejszy. Z powodzeniem można by było jednak ograniczyć się do wykorzystania jednej z nich.

Obie z przytoczonych bibliotek pozwalają na analizę tekstów w różnych językach, w tym języku polskim. Pozwalają na przeprowadzenie tokenizacji, czyli podziału tekstów na poszczególne słowa oraz lematyzacji, czyli zamiany każdego słowa na jego bazową formę. Można za ich pomocą znaleźć w tekście wszystkie nazwy własne poprzez analizę NER (Liu i in. 2022), czy też przypisać do każdego słowa nazwę części mowy, którą stanowi.

2.1.3 Biblioteka SQLParse

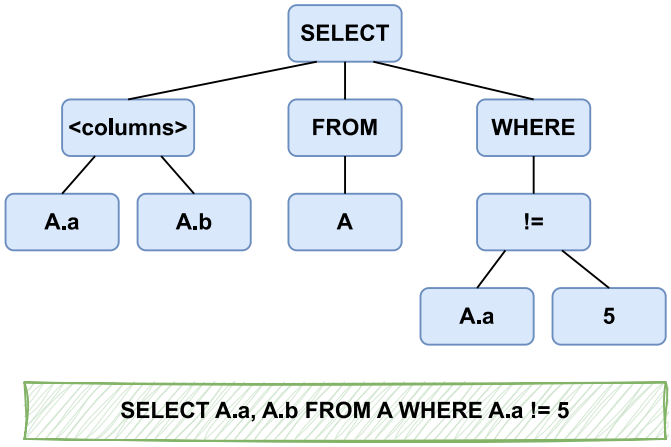
Biblioteka SQLParse[5] nie jest wyjątkowo rozbudowana pod względem oferowanych funkcji, lecz jej niezaprzeczalna popularność sugeruje, że doskonale wpasowała się w potrzeby deweloperów. Tak jak mówi nazwa, SQLParse pozwala na parsowanie zapytań SQL. Sprowadza się to do konwersji podanego jako wejście zapytania na poszczególne tokeny, takie jak słowa kluczowe SELECT, WHERE, nazwy kolumn i tabel oraz wartości. Każdy z wyprodukowanych przy tym tokenów posiada typ, mówiący co sobą reprezentuje. Biblioteka ta odegrała istotną rolę w procesie przygotowywania polskich zbiorów danych.

2.1.4 Biblioteka SQLGlot

Biblioteka SQLGlot[6] to naprawdę rozbudowane i wszechstronne narzędzie służące do pracy z instrukcjami SQL. Wspiera ponad 20 różnych dialektów i pozwala na transpilację, czyli konwersję instrukcji SQL pomiędzy nimi. Ponadto zapewnia szczegółową walidację i formatowanie. Wśród jej bardziej zaawansowanych funkcji znajduje się optymalizacja zapytań, a nawet symulowanie całego silnika bazodanowego, co pozwala na wykonywanie przez nią instrukcji.

[3] https://spacy.io
[4] https://stanfordnlp.github.io/stanza
[5] https://sqlparse.readthedocs.io
[6] https://sqlglot.com

Najbardziej pożądaną podczas realizacji niniejszej pracy funkcją okazało się parsowanie zapytań do drzew AST (ang. Abstract Syntax Tree) oraz możliwość dalszej pracy z nimi. Są to struktury, które pozwalają na reprezentację zapytań SQL, czy też instrukcji w dowolnym języku formalnym, w wygodnej do analizy i modyfikacji postaci. Drzewo AST dla przykładowego zapytania zostało przedstawione na rysunku 2.1.



Rys. 2.1: Drzewo AST dla przykładowego zapytania SQL

2.1.5 Biblioteka Streamlit

Biblioteka Streamlit[7] stanowi narzędzie do szybkiego tworzenia interaktywnych aplikacji internetowych opartych na danych. Została zaprojektowana z myślą o prostocie użycia, umożliwiając nawet początkującym programistom szybkie tworzenie interfejsów graficznych, bez konieczności dużej wiedzy na temat frontendu.

Streamlit oferuje intuicyjną składnię, która umożliwia tworzenie aplikacji poprzez strumieniowe przetwarzanie. Zapewnia integrację z wieloma bibliotekami do analizy danych i uczenia maszynowego. Jedną z kluczowych cech Streamlit jest automatyczne odświeżanie interfejsu użytkownika w odpowiedzi na zmiany w kodzie, co pozwala na szybką iterację i testowanie aplikacji. Dodatkowo biblioteka oferuje szeroki zakres interaktywnych elementów interfejsu, takich jak suwaki, pola wyboru, czy wykresy, które mogą być łatwo integrowane z analizowanymi danymi. W kontekście niniejszej pracy magisterskiej biblioteka Streamlit została wykorzystana w celu stworzenia aplikacji pozwalającej na przetestowanie stworzonych rozwiązań.

[7] https://streamlit.io

2.2 Visual Studio Code

`Visual Studio Code`[8] to wszechstronny edytor stworzony przez Microsoft. Zgodnie z wcześniej wspomnianą ankietą *2023 Developer Survey* (Stack Overflow 2023) jest on najczęściej wybieranym przez deweloperów środowiskiem programistycznym. Tym, co je wyróżnia, jest prostota przy jednoczesnej potędze płynącej z niespotykanej elastyczności. Dostępna jest bowiem ogromna liczba łatwych w instalacji rozszerzeń. Pozwalają one dostosować to środowisko niemal do każdego scenariusza wykorzystania.

Podczas realizacji pracy szczególnie użyteczne okazały się rozszerzenia dla języka Python dodające kolorowanie składni, sugerowanie kodu, debugowanie, czy też wsparcie dla interaktywnych notesów `Jupyter`. Poza tym intensywnie wykorzystywane było rozszerzenie do pracy z narzędziem `Docker`. Jeśli chodzi o podstawowe funkcje środowiska, to bardzo użyteczna okazała się integracja z systemem kontroli wersji oraz zaawansowane wyszukiwanie i zastępowanie.

2.3 Docker

`Docker`[9] to bardzo potężne i powszechnie wykorzystywane narzędzie, na którym opiera się znaczna część funkcjonującego oprogramowania. Umożliwia on tworzenie, zarządzanie i uruchamianie aplikacji w niemal całkowicie izolowanych środowiskach, tak zwanych kontenerach. Pozwalają one deweloperom na zapakowanie aplikacji z zależnościami i środowiskiem uruchomieniowym, co zapewnia spójność działania na różnych platformach.

Podczas realizacji niniejszego projektu `Docker` okazał się szczególnie użyteczny w kontekście uruchamiania istniejących modeli uczenia maszynowego. Część z nich była od razu gotowa do działania w kontenerze, a inne zostały do tego przystosowane. Okres czasu, w którym analizowane w dalszej części pracy rozwiązania powstawały, jest bowiem bardzo rozległy, więc zależności z których korzystają znacznie się różnią i trudno jest je ze sobą pogodzić. W przypadku prostych różnic w pakietach języka Python możliwe jest skorzystanie ze znacznie prostszego narzędzia `Conda`, lecz poziom izolacji tworzonych za jego pomocą środowisk jest znacznie ograniczony i jak się okazało często niewystarczający.

[8] `https://code.visualstudio.com`
[9] `https://www.docker.com`

2.4 Platforma Hugging Face

Hugging Face[10] to platforma dostarczająca narzędzia do tworzenia rozwiązań uczenia maszynowego. Zgromadzona jest na niej ogromna liczba pretrenowanych modeli i zbiorów danych, które może dodawać każdy. Dzięki temu zyskała ogromną popularność wśród projektów open source oraz badaczy. Nie jest to jednak wyłącznie platforma, ale także liczna społeczność, której celem jest zapewnienie równego dostępu do rozwiązań sztucznej inteligencji.

Korzystanie z zasobów udostępnionych przez Hugging Face jest niezwykle proste. Po wyszukaniu odpowiedniego modelu lub zbioru w interfejsie graficznym, wystarczy zwykle skopiować jego identyfikator i dodać do swojego skryptu jedną lub kilka linijek kodu. HF w szczególności skupia się na przetwarzaniu języka naturalnego i dostarcza do języka Python bibliotekę transformers. Daje ona możliwość konfigurowania, uruchamiania i trenowania modeli bazujących na architekturze transformerów (Vaswani i in. 2017), która daje obecnie najlepsze wyniki, jeśli chodzi właśnie o język naturalny.

[10] https://huggingface.co

3 Przegląd istniejących zbiorów

Zbiory danych w kontekście uczenia maszynowego stanowią fundamentalny element, wymagany do podejmowania wszelakich problemów. Mają bardzo duży wpływ na skuteczność każdego modelu i z tego powodu przed przystąpieniem do tworzenia polskich odpowiedników zostało poświęcone wiele wysiłku na przeanalizowanie zbiorów istniejących.

Na początku bieżącego rozdziału dokładnie przedstawiony zostanie zbiór `Spider`, ze szczególnym zwróceniem uwagi na jego format, którego polskie zbiory będą musiały przestrzegać. W kolejnej części nastąpi przegląd zbiorów do niego pokrewnych, czyli takich, które powstały na fundamencie baz danych `Spider` i również odegrały w niniejszej pracy istotną rolę. Na koniec nastąpi dogłębna analiza istniejących tłumaczeń zbioru `Spider`, aby dokonać świadomych decyzji podczas tworzenia tłumaczenia polskiego.

3.1 Angielski Spider

`Spider` to przede wszystkim zbiór danych przeznaczony do zadania `Text-to-SQL`, ale również wyzwanie z publicznie dostępnym rankingiem[11], które pozwala konkurować badaczom w tworzeniu coraz lepszych modeli oraz śledzić czynione postępy na przestrzeni czasu.

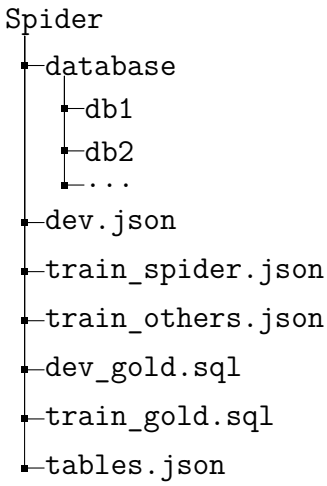
Na zbiór składa się 10 181 próbek, które są podzielone na trzy części: treningową, walidacyjną (zwaną również deweloperską) oraz testową. Ta ostatnia nie jest jednak powszechnie dostępna i sprawność swojego modelu można uzyskać na niej jedynie poprzez wysłanie go do wyzwania. Daje to pewność, że przypuszczalnie dobre wyniki wysyłanych modeli nie są skutkiem przypadkowego, ani celowego włączenia próbek testowych do zbioru treningowego, czyli zjawiska tak zwanego wycieku danych. Brak swobodnego dostępu do części testowej sprawia, że w praktyce część walidacyjna przejmuje jej rolę.

Próbki ze zbioru `Spider` opierają się na 200 bazach danych, które są bardzo różnorodne, ponieważ obejmują 138 domen, takich jak uczelnie, programy telewizyjne, czy administracja rządowa. W niektórych przypadkach ta sama tematyka poruszana jest przez kilka baz danych, co tłumaczy fakt, że jest ich więcej niż samych domen. Bardzo ważne jest to, że bazy są rozdzielone na poszczególne części zbioru. Między innymi żadna baza wykorzystywana w części treningowej nie jest używana w części walidacyjnej ani testowej. Wymusza to na uczonych za pomocą tego zbioru modelach posiadanie umiejętności uogólniania na nowe domeny.

[11] `https://yale-lily.github.io/spider`

3.1.1 Format zbioru

Wiele istniejących modeli wykorzystuje zbiór `Spider` ze względu na jego unikatowość i co za tym idzie, reprezentowany przez niego format stał się w pewien sposób standardem. Nie jest to prosty format tabelaryczny, jak to wygląda w postaci wielu innych zbiorów, lecz składają się na niego cztery jakościowo różne komponenty. Są to przykłady, poprawne zapytania SQL (ang. gold queries), schemat baz danych oraz same bazy. Nieco uproszczona struktura plików zbioru została przedstawiona na rysunku 3.1.



Rys. 3.1: Struktura plików zbioru `Spider`

Próbki (dev.json, train_spider.json, train_others.json)

Próbki (przykłady) przechowywane są w formacie JSON i podzielone zostały na trzy pliki. W `dev.json` znajdują się te odpowiadające zbiorowi walidacyjnemu, natomiast w plikach `train_spider.json` oraz `train_others.json` umieszczono próbki treningowe. Są one rozdzielone dodatkowo na dwa pliki, ponieważ w pierwszym twórcy zbioru umieścili swoje autorskie przykłady, natomiast w drugim te wyselekcjonowane z już istniejących zbiorów. Przykładowa próbka została przedstawiona na listingu 3.1.

```
{
 "db_id": "concert_singer",
 "question": "What is the average age of all singers from France?",
 "question_toks": ["What", "is", "the", "average", "age", "of", "all",
    ↪ "singers", "from", "France", "?"],
 "query": "SELECT avg(age) FROM singer WHERE country = 'France'",
 "query_toks": ["SELECT", "avg", "(", "age", ")", "FROM", "singer",
    ↪ "WHERE", "country", "=", "'France'"],
 "query_toks_no_value": ["select", "avg", "(", "age", ")", "from",
    ↪ "singer", "where", "country", "=", "value"],
 "sql": {}
}
```

Listing 3.1: Przykładowa próbka ze zbioru `Spider`

Znaczenie poszczególnych atrybutów próbki jest następujące:

- **`db_id`** – identyfikator bazy danych,
- **`question`** – pytanie w języku naturalnym,
- **`question_toks`** – pytanie podzielone na tokeny,
- **`query`** – zapytanie SQL,
- **`query_toks`** – zapytanie SQL podzielone na tokeny,
- **`query_toks_no_value`** – zapytanie SQL podzielone na tokeny, ale z zamaskowanymi wszystkimi wartościami, a dokładnie zastąpionymi tekstem `value`,
- **`sql`** – złożony obiekt JSON będący sparsowanym zapytaniem SQL, pozwalający na szybkie liczenie metryk i wydobywanie różnych fragmentów zapytania.

Poprawne zapytania SQL (`dev_gold.sql`, `train_gold.sql`)

Poprawne zapytania SQL, zgodnie z powyższym opisem, stanowią jeden z atrybutów próbek, lecz poza tym są również wyciągnięte do dwóch osobnych plików `dev_gold.sql` oraz `train_gold.sql`, które odpowiadają części walidacyjnej oraz treningowej. W każdej linii tych plików, jak przedstawiono na listingu 3.2, umieszczone jest poprawne zapytanie SQL oraz identyfikator bazy danych oddzielony od zapytania znakiem tabulacji.

```
SELECT count(*) FROM singer; concert_singer
SELECT Title FROM Cartoon WHERE Directed_by = "Ben Jones"; tvshow
SELECT petid, weight FROM pets WHERE pet_age > 1; pets_1
...
```

Listing 3.2: Fragment pliku `dev_gold.json` zawierającego poprawne zapytania SQL

Schemat (`tables.json`)

Plik `tables.json`, którego fragment został przedstawiony na listingu 3.3, opisuje strukturę każdej zawartej w zbiorze `Spider` bazy danych. Obejmuje to wskazanie nazw wszystkich tabel i kolumn, typów kolumn, kluczy podstawowych oraz relacji tworzonych przez klucze obce.

```
{
 "db_id": "concert_singer",
 "table_names_original": ["singer", "concert", "singer_in_concert", ...],
 "table_names": ["singer", "concert", "singer in concert", ...],
 "column_names_original": [
  [0, "Singer_ID"],
  [1, "Singer_ID"],
  [2, "concert_Name"],
  ...
 ],
 "column_names": [
  [0, "singer id"],
  [1, "singer id"],
  [2, "concert name"],
  ...
 ],
 "column_types": ["number", "number", "text", ...],
 "primary_keys": [0, ...],
 "foreign_keys": [[1, 0], ...]
 }
```

Listing 3.3: Fragment pliku `tables.json` opisujący schemat pojedynczej bazy danych

Znaczenie poszczególnych atrybutów obiektu opisującego schemat bazy jest następujące:

- `db_id` – identyfikator bazy danych,
- `table_names_original` – nazwy tabel występujących w bazie,
- `table_names` – naturalne odpowiedniki nazw tabel występujących w bazie,
- `column_names_original` – sekwencja dwuelementowych list zawierających kolejno numer porządkowy tabeli oraz nazwę zawartej w niej kolumny,
- `column_names` – tak jak wyżej, lecz z naturalnymi odpowiednikami nazw kolumn,
- `column_types` – typy danych posiadane przez wyżej wymieniane kolumny,
- `primary_keys` – numery porządkowe kolumn będących kluczami podstawowymi,
- `foreign_keys` – sekwencja dwuelementowych list zawierających kolejno numer porządkowy kolumny będącej kluczem obcym oraz numer porządkowy kolumny będącej powiązanym kluczem podstawowym.

Warto zwrócić uwagę na fakt, że w pliku schematu, oprócz oryginalnie występujących w bazie nazw tabel i kolumn, znajdują się również ich odpowiedniki w języku naturalnym. Oznacza to, że nazwy składające się z kilku słów i zapisywane oryginalnie za pomocą różnych konwencji, takich jak Camel Case (np. `playerName`), Snake Case (np. `player_name`), czy Pascal Case (np. `PlayerName`) są zamieniane na naturalną postać, czyli słowa odseparowane spacjami. Poza tym część oryginalnie skrótowych nazw jest rozwijana do bardziej zrozumiałej formy, dla przykładu `mid` zamieniane jest na `movie id`. Stanowi to dodatkową informację, która nie jest dostępna w samych bazach danych.

Bazy danych (`database/*`)

Ostatnim komponentem zbioru `Spider` jest katalog `database` zwierający szereg podkatalogów o nazwach odpowiadających identyfikatorom baz danych. Wewnątrz każdego z tych podkatalogów umieszczona jest baza danych `SQLite` oraz opcjonalnie dodatkowe pliki, takie jak skrypt SQL pozwalający zbudować daną bazę od zera. Większość baz, z nielicznymi wyjątkami, jest wypełniona danymi. Jest to istotne, ponieważ duża część nowoczesnych algorytmów opiera się na tych danych, by generowane zapytania SQL były dokładniejsze.

3.2 Angielskie zbiory pokrewne

Jedną z ważniejszych zasług zbioru Spider jest zgromadzenie pokaźnej liczby baz danych pochodzących z bardzo różnych domen. Było to trudne, ponieważ stanowią one informacje poufne dla wykorzystujących je firm i niewielka ich liczba jest dostępna w sieci. Nic więc dziwnego, że na fundamencie baz danych Spider powstało kilka kolejnych zbiorów. Doskonałym tego przykładem są zbiory CoSQL (T. Yu, Zhang, Er i in. 2019) oraz SParC (T. Yu, Zhang, Yasunaga i in. 2019), gdzie zachowano te same bazy, lecz stworzone zostały całkowicie nowe przykłady. Pojawiło się również odmienne podejście polegające na skonstruowaniu zbiorów pochodnych poprzez dokonanie w próbkach ze zbioru Spider pewnych ukierunkowanych modyfikacji, czego przykładem są zbiory Spider-Syn (Gan, Chen, Huang i in. 2021), Spider-DK (Gan, Chen i Purver 2021), czy Dr.Spider (Chang i in. 2023).

Jako że część ze wspomnianych powyżej zbiorów pokrewnych Spider odegrało istotną rolę w dalszej części pracy, to zostaną one pobieżnie opisane w poniższych sekcjach.

3.2.1 Spider-Syn

Spider-Syn (Gan, Chen, Huang i in. 2021) jest zbiorem danych powstałym ze zbioru Spider poprzez zmodyfikowanie próbek w taki sposób, aby ograniczyć dosłowne wymienianie w pytaniach nazw kolumn, tabel i wartości z baz danych. Odzwierciedla to scenariusz, w którym z interfejsu tekstowego do bazy korzysta osoba, która dokładnie jej nie zna, więc często posługuje się mimowolnie synonimami. Przykładowa próbka, w której słowo *singers* z oryginalnego zapytania zostało zastąpione bliskoznacznym słowem *musicians*, została przedstawiona na listingu 3.4. Stworzenie tego typu zbioru było istotne, ponieważ okazało się, że duża część istniejących algorytmów opiera się na powiązywaniu pytania z elementami baz danych poprzez znajdywanie dosłownych powtórzeń i w przedstawionym scenariuszu ich skuteczność radykalnie spada. Zbiór ten posiada część treningową oraz walidacyjną. Można w klasyczny sposób wykonać naukę modelu na części treningowej i ewaluację na części testowej, lecz również popularnym scenariuszem jest trening na zbiorze Spider i ewaluacja na obu częściach Spider-Syn.

```
{
  "db_id": "concert_singer",
  "SpiderQuestion": "What is the total number of singers?",
  "SpiderSynQuestion": "What is the total number of musicians?",
  "query": "SELECT count(*) FROM singer"
}
```

Listing 3.4: Przykład próbki ze zbioru Spider-Syn

3.2.2 Spider-DK

Zbiór danych Spider-DK (Gan, Chen i Purver 2021) (skrót od ang. Domain Knowledge) został zbudowany na podstawie części testowej Spider, w celu oceny tworzonych modeli pod kątem znajomości wiedzy domenowej. Składa się na niego skromna ilość próbek w liczbie 535, gdzie prawie połowa została wybrana ze zbioru Spider i przeniesiona bez zmian, a druga połowa powstała poprzez zmodyfikowanie przykładów tak, aby dodać wiedzę domenową. Przykładowa próbka została przedstawiona na listingu 3.5. Umieszczone w niej pytanie wymaga wywnioskowania, że pojęcia bycia najmłodszym i najstarszym są powiązane z datą urodzenia. Posiada ona atrybut type, który wskazuje jeden z pięciu przewidzianych typów wiedzy domenowej. W celu dokładniejszego zrozumienia można odwołać się do artykułu wprowadzającego zbiór Spider-DK, lecz w skrócie prezentują się one następująco:

- **Typ 1** – pomijanie wymieniania kolumn,
- **Typ 2** – proste wnioskowanie,
- **Typ 3** – synonimy wartości komórek,
- **Typ 4** – generowanie warunków na podstawie słów spoza wartości komórek,
- **Typ 5** – wiedza, która łatwo kłóci się z innymi dziedzinami.

```
{
  "type": 2,
  "db_id": "wta_1",
  "question": "Return names of all players sorted from oldest to youngest?"
  "query": "SELECT name FROM players ORDER BY birth_date",
}
```

Listing 3.5: Przykład próbki ze zbioru Spider-DK

3.2.3 SParC

SParC (T. Yu, Zhang, Yasunaga i in. 2019) jest zbiorem danych zawierającym sekwencje przeplatających się ze sobą pytań w języku naturalnym oraz instrukcji SQL otrzymanych w odpowiedzi. Obejmuje 4 298 sekwencji wiadomości, na które składa się łącznie ponad 12 tysięcy pojedynczych pytań. Są one ze sobą powiązane w obrębie konwersacji i wymagają wykorzystania informacji z poprzednich wiadomości, aby poprawnie sformułować żądane zapytanie SQL. Zbiór ten służy więc do nauki modeli rozwiązujących konwersacyjny wariant problemu Text-to-SQL. Przykładową konwersację z niego pochodzącą przedstawiono na rysunku 3.2. Widać na niej wyraźnie, że kolejne wiadomości bazują na kontekście wprowadzonym przez wiadomości wcześniejsze.

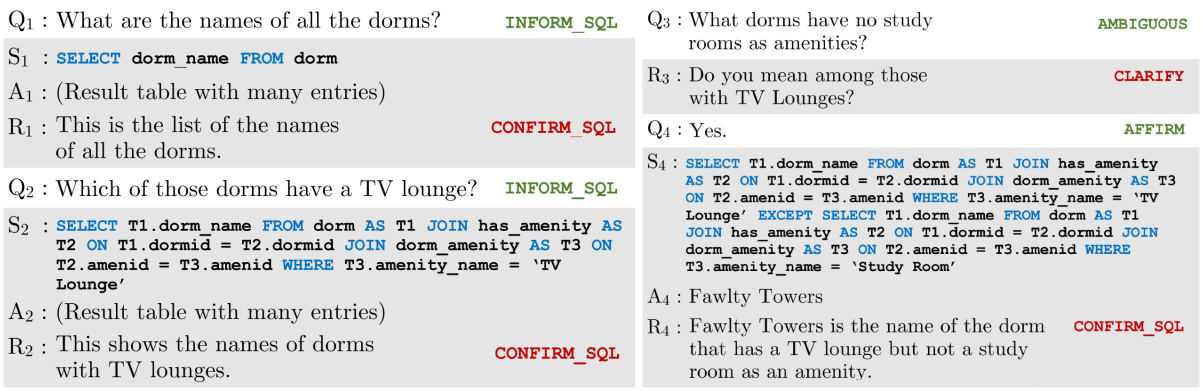
Q_1 : What is the customer id of the most recent customer?
S_1 : `SELECT customer_id FROM customers ORDER BY date_became_customer DESC LIMIT 1`
Q_2 : What is their name?
S_2 : `SELECT customer_name FROM customers ORDER BY date_became_customer DESC LIMIT 1`
Q_3 : How about for the first 5 customers?
S_3 : `SELECT customer_name FROM customers ORDER BY date_became_customer LIMIT 5`

Rys. 3.2: Przykładowa konwersacja ze zbioru SParC

Źródło: SParC (T. Yu, Zhang, Yasunaga i in. 2019)

3.2.4 CoSQL

CoSQL (T. Yu, Zhang, Er i in. 2019) jest zbiorem w znacznym stopniu podobnym do SParC, ponieważ zawiera dane o charakterze konwersacyjnym. Został do niego dodany jednak kolejny poziom złożoności, gdyż konwersacje nie stanowią już naprzemiennie powtarzających się pytań i instrukcji SQL, lecz powstały w drodze dalece swobodniejszych symulowanych rozmów pomiędzy zwykłymi użytkownikami a ekspertami SQL. Każdy dialog odtwarza realistyczny scenariusz eksploracji bazy danych, w którym użytkownik zadaje pytania, a ekspert stara się na nie odpowiedzieć, wyjaśniając niejednoznaczne kwestie, czy też informując o pytaniach bez odpowiedzi. Gdy pytania użytkownika dają się sformułować w języku SQL, to ekspert tego dokonuje i prezentuje wyniki w sposób pozwalający zachować naturalny przebieg interakcji. Przykładowa rozmowa, która to demonstruje, została przedstawiona na rysunku 3.3. Kompletny zbiór liczy 30 000 dialogów podzielonych na część treningową i testową. Łącznie zawierają 10 000 zapytań.



Rys. 3.3: Przykładowa konwersacja ze zbioru CoSQL

Źródło: CoSQL (T. Yu, Zhang, Er i in. 2019)

3.3 Tłumaczenia zbioru Spider

Tak jak zostało zaznaczone we wprowadzeniu, na chwilę obecną wydaje się istnieć pięć publicznie dostępnych zbiorów będących tłumaczeniami Spider. Cztery spośród nich dokonuje przekładu na konkretny język, a jeden zawiera tłumaczenia na kilka języków. Wspomniane zbiory jednojęzyczne to chiński CSpider, wietnamski ViText2SQL, rosyjski PAUQ oraz nie posiadający nazwy zbiór portugalski. MultiSpider to natomiast świeży zbiór, bo upowszechniony przez Microsoft w czasie pisanie niniejszej pracy, który zawiera niezależnie opracowane tłumaczenie chińskie, wietnamskie, niemieckie, francuskie, hiszpańskie oraz japońskie.

Poza publicznie dostępnymi zbiorami danych natrafiono również na wielojęzyczny zbiór XSpider (Shi i in. 2022), który zgodnie z artykułem zawiera przekłady na język hindi oraz farsi. Jego autorzy twierdzą w artykule, że wszystkie dane umieszczone zostały we wskazanym repozytorium, lecz jest ono całkowicie puste. Zmiana tego stanu rzeczy jest wątpliwa biorąc pod uwagę fakt, że artykuł ukazał się już ponad dwa lata temu.

Autorzy wymienionych zbiorów, pomimo wspólnego celu, podeszli do zadania na różne sposoby. Celem poniższych sekcji jest przedyskutowanie kluczowych różnic pomiędzy nimi. Jedną z najistotniejszych jest zastosowany rodzaj tłumaczenia. Rozbieżności zaobserwowano również w kwestii tłumaczenia schematu, zawartości baz danych oraz wartości w zapytaniach. Różnice te zostały zestawione w tabeli 3.1 i zostaną dokładnie omówione.

Zbiór	Rodzaj tłumaczenia	Tłumaczenie schematu	Tłumaczenie zawartości baz danych	Tłumaczenie wartości w zapytaniach
Chiński CSpider	Manualne	Nie	Nie	Tak
Wietnamski ViText2SQL	Manualne	Tak	Nie	Tak
Portugalski Brak Nazwy	Maszynowe	Tak	Nie	Nie
Rosyjski PAUQ	Manualne	Nie	Częściowo	Tak
Wielojęzyczny MultiSpider	Manualne	Tak	Nie	Obie wersje
Wielojęzyczny XSpider	——	Nie	——	——

Tabela 3.1: Zestawienie kluczowych różnic pomiędzy tłumaczeniami zbioru Spider. *Pozioma kreska oznacza brak informacji na dany temat.*

3.3.1 Rodzaj tłumaczenia

Tłumaczenie maszynowe i manualne to dwa oczywiste podejścia. Pierwsze sprowadza się do wykorzystania gotowych narzędzi, dostępnych najczęściej za pomocą webowych API, takich jak Google Cloud Translation API (Google 2024), czy DeepL (DeepL 2024). Dostępne są również rozwiązania działające offline, czego przykładem jest OpenNMT (Klein, Kim, Deng, V. Nguyen i in. 2018). Drugie podejście oznacza natomiast ręczne tłumaczenie każdej próbki przez człowieka. Oczywiście tłumaczenie ręczne może być wspomagane przez metodę maszynową, aby ograniczyć się do poprawiania tłumaczeń, zamiast pisania ich od początku.

Największą zaletą tłumaczenia maszynowego jest szybkie uzyskanie przetłumaczonego zbioru. Wiąże się to najczęściej z naliczeniem pewnych kosztów, o których należy wspomnieć, lecz mimo wszystko koszt ten jest niewspółmierny do ceny wynajęcia profesjonalnego ludzkiego tłumacza. Jest on również niewiele znaczący w stosunku do czasu, który trzeba by poświęcić, by dokonać tłumaczenia samemu.

Tłumaczenie maszynowe ma jednak istotne wady – pomimo że dostępne narzędzia stają się coraz lepsze, to wciąż nie dorównują człowiekowi. Sprawia to, że uzyskiwane zbiory danych są bezsprzecznie niższej jakości. Narzędzia te dokonując tłumaczenia zbioru Spider nie biorą pod uwagę wielu istotnych elementów, które ludzki adnotator by uwzględnił. Przykładem jest ignorowanie domeny podczas tłumaczenia pytania naturalnego. Ludzki tłumacz pytanie *list all*

parties w domenie politycznej przetłumaczy jako *zwróć wszystkie partie*. Tłumacz maszynowy natomiast może to przetłumaczyć jako *zwróć wszystkie imprezy*.

Dotychczasowe tłumaczenia zbioru Spider, z powodu ograniczeń metody maszynowej, w większości korzystają z ręcznego tłumaczenia zbioru. Jest to prawdą dla rosyjskiego PAUQ, wietnamskiego ViText2SQL, chińskiego CSpider oraz MultiSpider. Autorzy części z tych manualnie tłumaczonych zbiorów dla porównania dokonali również tłumaczenia maszynowego. Zestawienie wyników osiąganych przez modele trenowane na zbiorach tłumaczonych maszynowo i manualnie zostało przedstawione w tabeli 3.2. Wszystkie wyniki potwierdzają, że tłumaczenie manualne pozwala uzyskać lepsze rezultaty, aczkolwiek w przypadku względnie nowych modeli RAT-SQL oraz BRIDGE procentowy spadek skuteczności na zbiorach maszynowych nie jest tak znaczący, jak dla modeli wcześniejszych.

Zbiór	Model	Maszynowe	Manualne	Różnica
CSpider	C-ML	7,9	12,1	4,1
CSpider	W-ML	0,6	10,0	2,4
ViText2SQL	EditSQL (Vi-Syllable)	16,8	24,1	7,3
ViText2SQL	EditSQL (Vi-Word)	17,4	30,2	12,8
PAUQ	RAT-SQL	46,0	51,0	5,0
PAUQ	BRIDGE	49,0	52,0	3,0

Tabela 3.2: Wyniki modeli trenowanych na zbiorach tłumaczonych manualnie i maszynowo. *W trzech ostatnich kolumnach zamieszczono wyrażoną procentowo metrykę Exact Match Without Values, która zostanie dokładnie opisana w części 5.1. Wyższe wartości metryki oznaczają lepsze rezultaty.*

3.3.2 Tłumaczenie schematu

Kolejną kwestią dokonującą podziału wśród istniejących tłumaczeń zbioru Spider jest obrane podejście co do wynikowego języka schematu baz danych, czyli nazw tabel i kolumn – mogą być one tłumaczone, albo pozostawione bez zmian w języku angielskim. Drugie podejście jest uzasadnione faktem, że w praktyce, niezależnie od kraju, schemat baz danych jest często utrzymywany w języku angielskim. Ma to takie zalety jak ułatwienie pracy wielonarodowemu zespołowi, czy też uniknięcie problemu ze znakami diakrytycznymi, których wykorzystywanie może być niemożliwe lub utrudnione.

Fakt, że w praktyce nazwy tabel i kolumn zapisywane są w języku angielskim w swoich artykułach wyraźnie podkreślili autorzy tłumaczenia chińskiego, rosyjskiego oraz w pewnym stopniu zbioru XSpider i z uwagi na to postanowili schematu nie tłumaczyć. Twórcy zbioru wietnamskiego,

portugalskiego oraz `MultiSpider` postanowili jednak tłumaczenia dokonać, nie przedstawiając dla takiego podejścia specjalnego uzasadnienia.

Należy zauważyć, że reprezentowanie pytań i schematu baz danych w dwóch różnych językach znacząco komplikuje rozwiązywany problem. W wariancie ze wszystkimi komponentami w tym samym języku dość łatwo jest znaleźć powiązania pomiędzy nimi, ponieważ wystarczy przeanalizować ich podobieństwo jako łańcuchów znaków. W przeciwnym przypadku to jednak nie wystarczy – wykorzystywany model musi przejawiać dodatkowo pewne cechy tłumacza.

Podsumowując, pozostawienie schematu baz danych w języku angielskim jest uzasadnione z praktycznego punktu widzenia, jednak dodatkowo komplikuje zadanie. Zbiór danych z przetłumaczonym schematem, chociaż w dużym stopniu ignoruje aspekt praktyczny, stanowi lepszy odpowiednik oryginalnego zbioru `Spider` i pozwala na wykonywanie względem niego bardziej sprawiedliwych porównań.

3.3.3 Tłumaczenie zawartości baz danych

Kolejną niespójnością, jaką można zaobserwować pomiędzy istniejącymi tłumaczeniami zbioru `Spider`, jest kwestia tłumaczenia zawartości baz danych. Dokonanie takiego tłumaczenia ma istotne zalety przedstawione w kolejnych akapitach, jednak jest to zadanie problematyczne, przez co nie wszyscy autorzy zbiorów się go podjęli.

Większość nowych podejść do problemu generowania SQL analizuje znajdujące się w bazach danych rekordy, aby zwiększyć swoją dokładność. Przykładowo użytkownik systemu może poprosić o zwrócenie populacji Polski, a dwie możliwe odpowiedzi mogłyby zawierać fragmenty `WHERE country = 'PL'` oraz `WHERE country = 'polska'`. Widać tutaj, że schemat bazy nie mówi wszystkiego i bez dostępu do jej zawartości obie odpowiedzi są równie dobre, choć tylko jedna może być poprawna. W celu wytrenowania najlepszych modeli przygotowany zbiór powinien zawierać więc wypełnione bazy danych, a większość istniejących rozwiązań, korzystających z zawartości baz, zakłada, że jest ona w tym samym języku co pytania.

Drugim celem wartości w bazach danych jest umożliwienie ewaluacji opracowanych algorytmów `Text-to-SQL` za pomocą metryki `Execution Accuracy`. Została ona opisana dokładnie w późniejszej części pracy, w sekcji 5.1. Jej skrótowa zasada działania polega na wykonaniu prawidłowych zapytań oraz zapytań wygenerowanych i uznaniu za poprawne tych, które zwróciły te same wyniki. Przetłumaczenie wartości w zapytaniach SQL bez tłumaczenia zawartości baz

danych skutkuje tym, że wykonywane zapytania często nie zwracają żadnych wyników, a co za tym idzie, wiele nieprawidłowo wygenerowanych zapytań jest uznawanych omyłkowo za prawidłowe. Podsumowując, różne języki wartości w zapytaniach SQL i wartości w bazach danych skutkują upośledzeniem metryki `Execution Accuracy` i zmniejszeniem jej użyteczności.

Modyfikacji zawartości baz danych podjęli się jedynie autorzy tłumaczenia rosyjskiego. Nie tłumaczyli jednak wszystkich wartości, ale tylko te, które wystąpiły w przynajmniej jednym zapytaniu. Powodem, dla którego pozostałe zbiory obyły się bez tłumaczenia, jest zapewne duża liczba danych zawartych w bazach i relatywnie niewielkie korzyści wynikające z ich tłumaczenia. Ciekawy pomysł został zastosowany w zbiorze portugalskim oraz `MultiSpider`, gdzie zawartość baz danych nie jest tłumaczona, ale wartości w zapytaniach również. Takie podejście pozwala uniknąć obu zarysowanych w powyższych akapitach problemów, ale wydaje się mało praktyczne – zakłada scenariusz w których w bazach danych o przykładowo portugalskim schemacie przechowywane są dane angielskie.

4 Tworzenie polskich zbiorów

W ostatnich zwłaszcza latach najwięcej uwagi poświęca się nie modelom, lecz odpowiedniemu przygotowaniu i jakości zbiorów danych. Przykładem tego są wydane w ostatnim czasie artykuły *Textbooks are all you need* (Gunasekar i in. 2023) oraz *Orca 2: Teaching small language models how to reason* (Mitra i in. 2023), które demonstrują jak dużą poprawę można uzyskać bez powiększania i komplikowania wykorzystywanego modelu, lecz poprzez stworzenie wysokiej jakości zbioru danych. Jako że dla zadania `Text-to-SQL` na moment pisania niniejszej pracy nie istnieją żadne zbiory danych w języku polskim, a także ze względu na duże ich znaczenie, wiele uwagi zostało poświęcone tej kwestii.

Na początku bieżącego rozdziału omówione zostaną przyjęte założenia dotyczące tworzenia polskich zbiorów, a następnie proces ten zostanie krok po kroku opisany. W pierwszej kolejności przedstawiony będzie sposób wstępnego przetworzenia zbiorów angielskich, a następnie skrypt służący do generowania zbiorów polskich oraz sposób dokonywania poszczególnych tłumaczeń. Na koniec opracowane zbiory zostaną podsumowane i przeanalizowane.

4.1 Przyjęte założenia

Po zrozumieniu różnic między istniejącymi tłumaczeniami zbioru `Spider` przystąpiono do zdefiniowania założeń na rzecz opracowania tłumaczeń polskich. Przyjęte założenia obejmują wykorzystanie tłumaczenia maszynowego, generowanie finalnych zbiorów w sposób skryptowy, tłumaczenie dodatkowo zbiorów pokrewnych oraz zachowanie oryginalnej zawartości baz danych. Każde z tych założeń zostanie uzasadnione w osobnej sekcji.

4.1.1 Tłumaczenie maszynowe

Pomimo wskazanej w sekcji 3.3.1 dominacji tłumaczenia manualnego nad maszynowym postanowiono wykorzystać to ostatnie. Przyczyn tej decyzji jest kilka. Przede wszystkim w realizację tłumaczenia aktywnie zaangażowana jest tylko jedna osoba, w odróżnieniu od wcześniejszych prac, w których w tym procesie uczestniczyło ich kilka, w tym nawet profesjonalni tłumacze. Drugą przesłanką jest ograniczony czas, ze względu na pracę magisterską, w ramach której niniejszy temat jest realizowany. Ostatecznie jest to zadanie żmudne, a zbiór maszynowy, pomimo niższej jakości, także pozwoli, a nawet da więcej czasu, na wykonanie eksperymentów.

4.1.2 Skryptowe generowanie zbiorów

Wszystkie dotychczasowe tłumaczenia zbioru Spider sprowadzają się jedynie do udostępnienia samych danych, czyli zmodyfikowanej wersji zbioru, bez żadnych dodatkowych skryptów. Wydaje się to wystarczające i dla większości zastosowań w rzeczywistości jest. Taki zbiór stanowi bardzo dobry benchmark służący do porównywania różnych algorytmów, ponieważ nie ma żadnych niedomówień w kwestii jego zawartości.

W niniejszej pracy zaproponowano niespotykane, a przynajmniej nie opisane, we wcześniejszych pracach podejście, które polega na skryptowym generowaniu zbiorów w żądanej konfiguracji. Na początku zostaną one rozbite na elementarne składniki, czyli pytania, zapytania SQL oraz nazwy tabel i kolumn. Następnie, po dokonaniu tłumaczenia indywidualnych elementów, będzie można przeprowadzić ich skryptową syntezę w kompletny zbiór, wybierając przy tym żądany język pytań, język zapytań, czy jeden z przypuszczalnie kilku wariantów nazewnictwa tabel i kolumn.

Zdecydowano się na wykorzystanie zarysowanej metody przede wszystkim dlatego, że zdekomponowanie zbioru znacznie ułatwia tłumaczenie. Chociażby nazwy tabel i kolumn występują zarówno w pliku schematu i próbkach, więc praca na zbiorze w pierwotnej postaci wymagałaby wprowadzania podobnych zmian w kilku miejscach, co jest niewygodne i wrażliwe na błędy. Przypuszcza się, że autorzy wcześniejszych tłumaczeń również mogli wykorzystać podobną technikę, w szczególności ci, którzy dokonywali tłumaczenia schematu. W ich artykułach nie natrafiono jednak na przesłanki mówiące, że tak właśnie było i przedstawiona w dalszej części pracy metoda jest własnym pomysłem autora niniejszej pracy.

Korzystając z opisanej metody można stworzyć kilka mapowań starych nazw tabel i kolumn na nowe i do syntezy wykorzystać to, które w danym przypadku wydaje się najlepsze. Przykładowo bazy mogą zawierać nazwy angielskie, polskie, czy też polskie bez znaków diakrytycznych. Jest to poza zakresem pracy, lecz skrypt, który zostanie opracowany, w dużym stopniu ułatwi dokonywanie tłumaczeń na inne języki, bo wszystkie komponenty zbioru `Spider` zostały już rozplątane. Generalnie taka strategia jest bardzo elastyczna i otwiera drogę na nowe eksperymenty.

4.1.3 Tłumaczenie zbiorów pokrewnych

Oprócz przetłumaczenia zbioru `Spider` postanowiono dokonać tego również dla czterech zbiorów pokrewnych przedstawionych wcześniej w sekcji 3.2. Naturalnym powodem jest chęć zdobycia

większej liczby próbek w nadziei, że pozwoli to finalnie wytrenować lepszej jakości modele. Dodatkowe zbiory mogą zostać wykorzystane także do testów. W szczególności Spider-Syn oraz Spider-DK pozwolą na testowanie tworzonych algorytmów pod kątem odporności na synonimy oraz znajomości wiedzy domenowej.

Przetłumaczenie dodatkowych zbiorów jest ułatwione, gdyż ich format jest w dużym stopniu zgodny z formatem zbioru Spider. Poza tym korzystają z tych samych baz danych, co pozwala uniknąć dodatkowego tłumaczenia nazw schematu. Całość upraszcza wykorzystanie tłumaczenia maszynowego oraz skryptowej syntezy. Nie znaleziono informacji na temat wcześniejszych prób tłumaczenia tych zbiorów, więc niniejsza praca być może jest pierwsza.

4.1.4 Oryginalna zawartość baz danych

Zawartość baz danych postanowiono pozostawić bez zmian – ewentualnemu tłumaczeniu podlegać będzie jedynie ich schemat. Powodem tego jest duża ilość znajdujących się tam informacji, których tłumaczenie jest kłopotliwe, a także względnie niewielkie korzyści z tego płynące. Jest to jednocześnie zgodne z większością istniejących podejść do tworzenia przekładów zbioru Spider. Modele trenowane na nich wciąż są w stanie osiągać wysokie wyniki, co potwierdza słuszność takiego podejścia.

Rozważano dokonanie tłumaczenia z wykorzystaniem narzędzi działających offline, by uniknąć wysokich kosztów, lecz nie rozwiązuje to wszystkich problemów. W bazach znajduje się wiele informacji, które należałoby spolszczyć, a tłumacze maszynowe tego nie dokona, czego przykładami są imiona, nazwiska, adresy i skrótowe nazwy. Poza tym wartości występują również w zapytaniach SQL i konieczna by była ostrożność, aby odpowiednie wartości z baz oraz zapytań zostały zgodnie przetłumaczone.

4.2 Przygotowanie zbiorów angielskich

Poza angielskim zbiorem Spider, którego przetłumaczenie bez wątpienia jest najistotniejsze, powzięto również tłumaczenie zbiorów Spider-Syn, Spider-DK, SParC oraz CoSQL. Ich struktura nieco odbiega od zbioru Spider, więc konieczne okazało się wykonanie dodatkowych kroków, które zostały w niniejszej części podsumowane. Niektóre z nich są konieczne, tak jak zamiana zbiorów z kontekstowych na bezkontekstowe, a inne opcjonalne, jak przeprowadzenie deduplikacji. Sprawiają one, że tłumaczenia owych dodatkowych zbiorów nie stanowią ideal-

nych odpowiedników oryginałów. W różnym stopniu, ale należy traktować je jako nowe zbiory i unikać zestawiania osiąganych na nich rezultatów z angielskimi odpowiednikami.

4.2.1 Konwersja zbiorów kontekstowych na bezkontekstowe

W ramach realizowanego tematu rozważany jest problem tłumaczenia bezkontekstowego, co jest pozornie sprzeczne z wykorzystaniem zbiorów SParC oraz CoSQL. Nie zawierają one bowiem prostego szeregu pytań i odpowiadających im zapytań SQL, lecz dodatkowo porządkują je w konwersacje, gdzie kolejne wiadomości bazują na znajomości poprzednich. Zauważono jednak, że zbiory kontekstowe mogą zostać przekształcone na bezkontekstowe poprzez przefiltrowanie polegające na wybraniu z każdej konwersacji jedynie pierwszego pytania i odpowiedzi. Nie istnieje wówczas żadna historia wiadomości wcześniejszych, więc takie przykłady nie posiadają kontekstu. Powstałe w ten sposób zbiory znacznie różnią się od oryginalnie kontekstowych SParC oraz CoSQL, więc całkowicie nieuzasadnionym jest porównywanie ze sobą osiąganych na nich wyników.

4.2.2 Modyfikacja pytań

Część pytań wewnątrz zamienionych na bezkontekstowe zbiorów SParC oraz CoSQL wyraźnie odróżnia się od reszty. Zawierają bowiem wyrażenia charakterystyczne dla konwersacji, takie jak *Hi*, *Hello*, *How are you*. Przykładem jest pytanie „Hello, how many total documents are there?”. Wymienione wyrażenia są dla rozważanego problemu niepożądane, gdyż nie stanowią typowych zapytań, jakie użytkownik wprowadziłby do systemu, wiedząc, że zwraca jedynie zapytania SQL. Z tego względu znaleziono takie przypadki za pomocą wyrażeń regularnych oraz funkcji wyszukiwania, dostępnej w wykorzystywanym edytorze i usunięto z pytań niechciane początki.

4.2.3 Deduplikacja w obrębie zbiorów

Za zduplikowane zostały uznane przykłady, które posiadają takie same bazy danych oraz znormalizowane angielskie pytania (przekonwertowane na małe litery i bez niepotrzebnych białych znaków). Deduplikacji w obrębie zbiorów dokonano poprzez zwyczajne usunięcie występujących kilkukrotnie próbek. Okazało się, że znaleziono takie w każdym ze zbiorów, poza Spider-DK. 25 przypadków zostało zlokalizowanych nawet w samym zbiorze Spider, lecz w nim modyfikacji nie wprowadzano, bo ze względu na jego wysoką renomę chciano, by próbki w polskim

wariancie dokładnie mu odpowiadały. Najwięcej duplikatów zostało znalezionych w zbiorze SParC. Widocznie w tym oryginalnie kontekstowym zbiorze wiele konwersacji rozpoczynało się od tych samych wiadomości i różnice następowały dopiero później.

4.2.4 Deduplikacja pomiędzy zbiorami

Kolejnym problemem okazały się duplikaty występujące pomiędzy zbiorami. W większości występowały one między Spider a całą resztą, ponieważ Spider powstał jako pierwszy. Widać to doskonale na rysunku 4.1a, gdyż największa liczba duplikatów znajduje się w wierszu oraz kolumnie odpowiadającej właśnie zbiorowi Spider. Dużą liczbą powtórzeń wyróżnia się w szczególności para zbiorów Spider oraz Spider-Syn, ponieważ w tym ostatnim autorzy umieszczali oryginalną próbkę, jeżeli nie udało im się wymyślić rozsądnego synonimu. Ostatecznie całą duplikację ze zbiorem Spider postanowiono usunąć. Jedynym wyjątkiem są powtórzenia pomiędzy parą Spider-DK oraz Spider, które pozostawiono, ponieważ Spider-DK ma charakter czysto walidacyjny i 206 znalezionych duplikatów to nie przypadkowe próbki, lecz precyzyjnie wyselekcjonowane ze zbioru Spider przykłady zawierające wiedzę domenową. Pozostawioną w zbiorach duplikację, po usunięciu części próbek, przedstawiono na rysunku 4.1b.

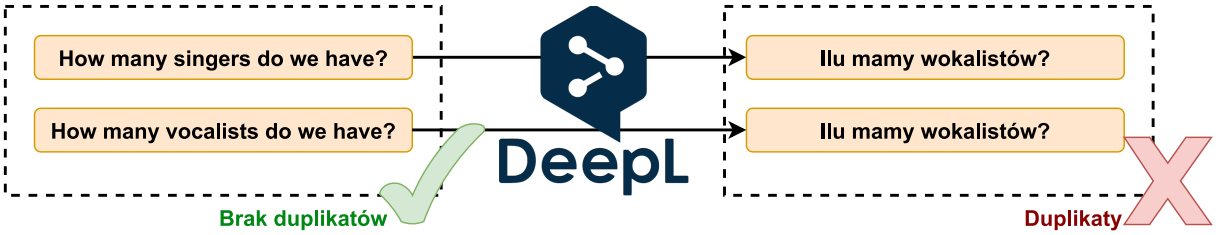
(a) Przed deduplikacją

(b) Po deduplikacji

Rys. 4.1: Diagramy ilustrujące liczbę duplikatów pomiędzy zbiorami. *Na osiach umieszczono nazwy zbiorów wraz z liczbą próbek, a wartości w komórkach macierzy wskazują liczbę duplikatów pomiędzy daną parą zbiorów.*

4.2.5 Dodatkowa deduplikacja tłumaczeń

Trzeba zauważyć, że opisane powyżej usuwanie duplikatów bazuje na angielskich wersjach pytań, co wydaje się wystarczające, ponieważ jeżeli pytania angielskie się różnią, to ich polskie tłumaczenia również powinny się różnić. W przypadku próbek zawartych w zbiorze `Spider-Syn` jest jednak często inaczej. Dla przypomnienia, zawiera on próbki z oryginalnego zbioru `Spider`, lecz z wybranymi słowami zamienionymi na synonimy. Okazuje się, że te synonimy oraz bazowe słowa, od których pochodzą, tłumaczone są niejednokrotnie przez tłumacza maszynowego na te same polskie wyrażenia, co skutkuje duplikatami w liczbie 1350. Proces ich powstawania został zilustrowany na rysunku 4.2. Aby pozbyć się tak powstałej duplikacji postanowiono usunąć niepotrzebne próbki ze zbioru `Spider-Syn`. Działanie to, w odróżnieniu od wcześniej opisanych, musiało się odbyć już po dokonaniu tłumaczenia na język polski.



Rys. 4.2: Sposób powstawania duplikacji pomiędzy `Spider` i `Spider-Syn` w wyniku tłumaczenia

4.3 Skryptowe generowanie zbiorów

Skryptowe generowanie zbiorów danych jest ważnym elementem niniejszej pracy, który nie został w istniejącej literaturze napotkany. Z tego względu proces ten zostanie w poniższych sekcjach dokładnie opisany. Na początku przedstawione będą dane źródłowe, które niezbędne są w celu przeprowadzenia generacji. W dalszej kolejności omówiony zostanie algorytm służący do mapowania starych nazw schematu na nowe, a na koniec sposób rekonstrukcji redundantnych informacji, które zostały wcześniej usunięte w celu uproszczenia przetwarzania.

4.3.1 Dane źródłowe

Przed przystąpieniem do tworzenia skryptu generującego zbiory konieczne okazało się zastanowienie nad danymi źródłowymi, których on potrzebuje. Zdecydowano się rozbić je na dwie grupy: próbki, zawierające pytania i zapytania SQL w różnych językach oraz mapowania nazw schematu, zawierające kompletne odwzorowania oryginalnych nazw tabel i kolumn na nowe.

Informacje te postawiono przechowywać w osobnych plikach JSON, których format zostanie dalej opisany. Taka dekompozycja umożliwia dodawanie kolejnych zbiorów poprzez tworzenie nowych plików próbek i umożliwianie innych sposobów tłumaczenia nazw tabel i kolumn poprzez tworzenie kolejnych plików mapowania.

Plik próbek

Plik próbek zawiera listę obiektów JSON, z których każdy reprezentuje konkretny przykład. Jeden z nich przedstawiono na rysunku 4.1. Zawiera jedynie niezbędne informacje, czyli nazwę bazy danych oraz alternatywne warianty pytań i zapytań SQL. Posiada angielskie i polskie ich wersje, ponieważ takie są istotne dla realizacji niniejszej pracy. Nic nie stoi jednak na przeszkodzie, aby dodać więcej języków, gdyż zarówno format omawianego pliku, jak i reszta komponentów, zostało na taką okoliczność przygotowane.

```
{
 "db_id": "local_govt_in_alabama",
 "question": {
  "en": "How many participants have the type 'Organizer'?",
  "pl": "Ilu uczestników posiada typ 'Organizator'?"
 },
 "query": {
  "en": "SELECT count(*) FROM participants WHERE type_code='Organizer'",
  "pl": "SELECT count(*) FROM participants WHERE type_code='Organizator'"
 }
}
```

Listing 4.1: Obiekt JSON z pliku próbek reprezentujący jeden przykład

Warto zwrócić uwagę na fakt, że próbki w pełnym zbiorze posiadają znacznie więcej atrybutów niż te przedstawione powyżej. Zawierają dodatkowo pytania i zapytania SQL podzielone na tokeny oraz sparsowane zapytania. Są to jednak w dużej części informacje redundantne, które można z powodzeniem odtworzyć na podstawie skromnych informacji przedstawionych powyżej i jest to rzeczywiście jedna z funkcji realizowanych przez skrypt generujący. Odtworzenie danych redundantnych niestety wymaga napisania dodatkowego kodu oraz dodaje narzut czasowy do procesu generacji. Z drugiej strony mocno upraszcza dane źródłowe, co jest znacznie ważniejsze biorąc pod uwagę cel, którym jest dokonanie tłumaczenia.

Plik mapowania nazw

Aby umożliwić dokonywanie różnych tłumaczeń schematów baz danych, czyli nazw tabel i kolumn, opracowany został format pliku przedstawiony na listingu 4.2. Zawiera on informacje pozwalające na dokonanie mapowania oryginalnych nazw na dowolne inne. Jest to plik JSON o kilku stopniach zagnieżdżenia. Na najwyższym poziomie jest słownikiem, który dla każdej nazwy bazy danych zawiera listę tabel, a każda tabela listę kolumn. Wszystkie obiekty reprezentujące tabele i kolumny posiadają nazwy źródłowe oraz docelowe, na które powinny zostać przetłumaczone. Jak już wiadomo, mają one dwa rodzaje nazw, które zostały tutaj uwzględnione: oryginalnie występujące w bazach danych oraz odpowiedniki w języku naturalnym.

```
{
    "concert_singer": [
        {
            "orig_src": "singer_in_concert",
            "orig_tgt": "piosenkarz_na_koncercie",
            "name_src": "singer in concert",
            "name_tgt": "piosenkarz na koncercie",
            "columns": [
                {
                    "orig_src": "Singer_ID",
                    "orig_tgt": "Piosenkarz_ID",
                    "name_src": "singer id",
                    "name_tgt": "piosenkarz id",
                },
                <kolejne kolumny>
            ]
        },
        <kolejne tabele>
    <kolejne bazy>
}
```

Listing 4.2: Fragment opracowanego pliku mapowań nazw schematu

Należy zwrócić uwagę na to, że zaproponowany format pozwala, aby dana nazwa kolumny była tłumaczona na różne sposoby w zależności od tabeli i bazy, w której się znajduje. Podobnie ta sama nazwa tabeli może być tłumaczona na różne sposoby w zależności od zawierającej ją bazy

danych. Jest to celowy zabieg i wymagany w celu umożliwienia prawdziwie wysokiej jakości tłumaczenia. Rozważmy dla przykładu tabelę o nazwie `department`. W bazie dotyczącej biznesu prawdopodobnie powinna zostać przetłumaczona jako `dział`, natomiast w domenie uniwersyteckiej jako `wydział`. Przyjęcie przedstawionej struktury pliku mapowania nazw umożliwiło dokonanie tego typu kontekstowych tłumaczeń, co zostało wykorzystane w dalszej części pracy.

4.3.2 Zmiana nazw tabel i kolumn

Zmiana nazw tabel i kolumn zgodnie z opisanym wyżej plikiem mapowania to dla opracowanego procesu generacji zbiorów najważniejszy punkt. Sprowadza się do dokonania modyfikacji w dwóch miejscach. Podmiana nazw w samych bazach danych `SQLite` jest prostszą częścią. Znacznie trudniejsze okazuje się dokonanie zmian w zapytaniach. Powstały one przecież na bazie oryginalnych nazw tabel i kolumn, które należy wychwycić i podmienić na nowe.

Zmiana nazw w bazach danych

Zmodyfikowanie nazw tabel i kolumn w bazach danych nie stanowi dużego wyzwania, ponieważ wystarczy wykonać na każdej z nich serię instrukcji SQL typu `ALTER TABLE`. Ułatwia to fakt, że język Python posiada w ramach biblioteki standardowej pakiet pozwalający na pracę z bazami `SQLite`. Przystępując do tego należy jednak mieć pewność, że zastosowanie danego mapowania nie tworzy konfliktujących ze sobą pod względem nazw elementów, bo wówczas operacja się nie powiedzie. Zadbano o ten aspekt podczas opisanego w dalszej części tłumaczenia pliku mapowań nazw. Podczas implementacji tego etapu zauważono, że należy ignorować tabele `sqlite_sequence`, ponieważ jest to specjalna tabela i modyfikacja jej nazwy nie jest możliwa.

Zmiana nazw w zapytaniach

Dokonanie podmiany nazw tabel i kolumn w zapytaniach SQL stanowi najtrudniejszy etap w całym procesie generowania zbiorów. Przyczyną tego jest przyjęte podejście polegające na tym, że dana kolumna może być przetłumaczona na różne sposoby w zależności od tabeli, w której się znajduje. W przypadku modyfikacji baz danych było oczywistym, do której tabeli należy każda kolumna, bo wynika to z ich jasno zdefiniowanej struktury. Zapytania SQL są natomiast w gruncie rzeczy zwykłym tekstem, więc wydobycie z nich kolumn oraz ustalenie powiązanych tabel stanowi niemałe wyzwanie.

Najłatwiejsze podejście do przedstawionego problemu, ale posiadające istotną słabość, wykorzystuje parsowanie zapytań do postaci drzew AST z wykorzystaniem przedstawionej wcześniej biblioteki SQLGlot. Na poziomie drzewa można łatwo dokonać modyfikacji nazw, bo ustalenie przynależności kolumn do tabel jest znacznie prostsze ze względu na jego hierarchiczną strukturę. Po modyfikacji drzewa należy zrekonstruować na jego podstawie zapytanie SQL, do którego wprowadzone zmiany się przeniosą. Jak zostało sprawdzone, taka strategia pozwala na skuteczne przetłumaczenie nazw schematu, jednak wspomnianą słabością jest to, że przetworzone zapytania różnią się delikatnie od oryginałów pod względem formatowania oraz struktury. Powodem są następujące po sobie etapy parsowania i rekonstrukcji z wykorzystaniem biblioteki SQLGlot, które zachowują jedynie znaczenie zapytań, pozwalając sobie przy tym na delikatne modyfikacje zapisu. Zostało to przedstawione na rysunku 4.3. Przetłumaczone zapytania odbiegają więc nadmiernie od pierwotnych i porównywanie stworzonego tak zbioru z oryginalnym mogłoby być kwestionowane. W szczególności wiele algorytmów opiera się na specyficznym dla poszczególnych zbiorów formacie zapytań i jego modyfikacja mogłaby doprowadzić do problemów z uruchomieniem części modeli. Z przytoczonych powodów takie podejście zostało porzucone.

```sql
SELECT avg(bikes_available)
FROM status
WHERE station_id NOT IN
    (SELECT id
     FROM station
     WHERE city  =  "Palo Alto")
```

(a) Zapytanie pierwotne

```sql
SELECT AVG(bikes_available)
FROM status
WHERE NOT station_id IN
    (SELECT id
     FROM station
     WHERE city = "Palo Alto")
```

(b) Zapytanie zmodyfikowane

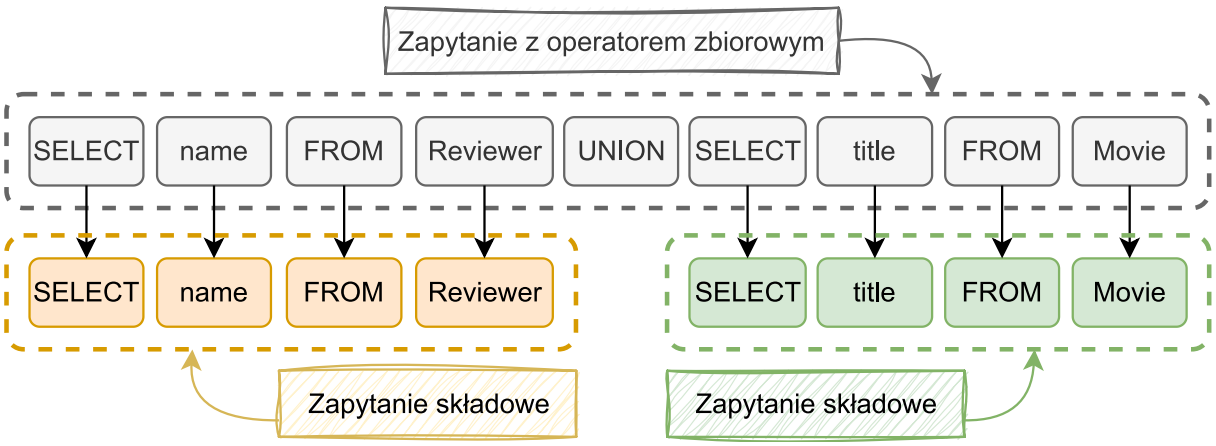
Rys. 4.3: Przykład drobnych modyfikacji wprowadzanych przez bibliotekę SQLGlot. *Zapytanie po prawej stronie powstało przez sparsowanie lewego do drzewa AST i powrót do postaci SQL.*

W celu przetłumaczenia nazw w zapytaniach, bez niepotrzebnego wpływania na strukturę, zdecydowano się ostatecznie zrobić to na niższym poziomie. Opracowany został skomplikowany algorytm, który dokonuje tokenizacji zapytań, a następnie analizuje wszystkie tokeny po kolei i jeżeli wykryje nazwę tabeli bądź kolumny, to podmienia ją na nową. Aby sprawdzić, czy dany token jest nazwą, dokonywano parsowania zapytania do drzewa AST, następnie modyfikowano token na inny i ponownie parsowano zapytanie do AST. Porównując ze sobą tak powstałe drzewa, można ustalić, czy zmodyfikowanym tokenem jest nazwa tabeli, nazwa kolumny, czy żadne z powyższych. W przypadku nazw kolumn konieczne jest dodatkowo ustalenie przynależności do tabeli. Wyróżniono tutaj trzy przypadki:

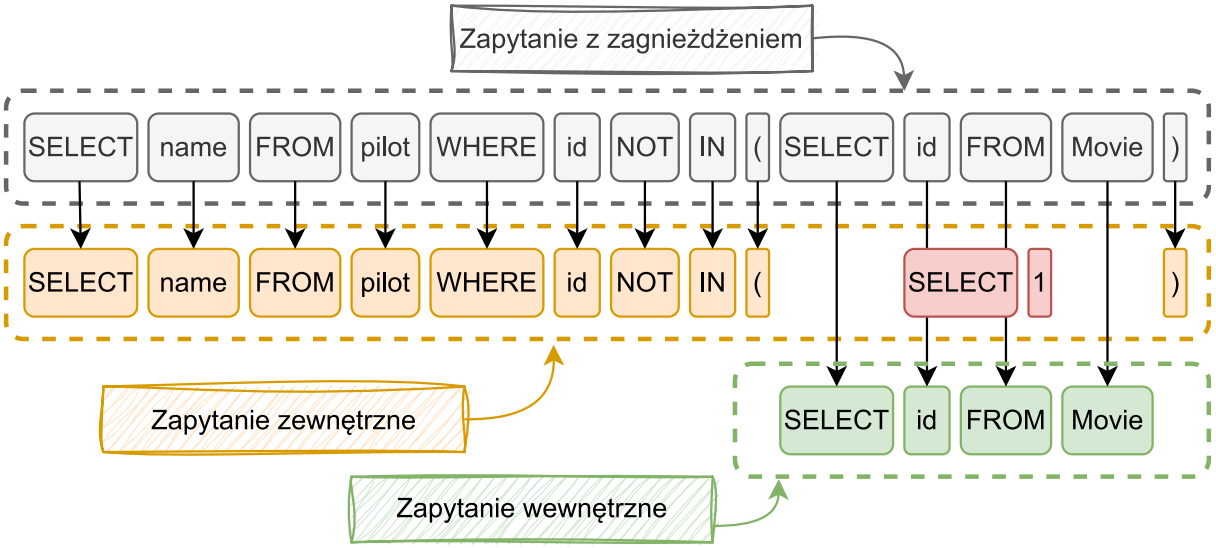
1. Nazwa kolumny poprzedzona nazwą tabeli (`SELECT order.id FROM order`)
2. Nazwa kolumny poprzedzona aliasem (`SELECT T1.id FROM order as T1`)
3. Nazwa kolumny bez tabeli (`SELECT id FROM order`)

Pierwszy scenariusz jest najprostszy, ponieważ nazwa tabeli jest jawnie podana i nie ma co do niej wątpliwości. Drugi przypadek jest bardziej skomplikowany, gdyż zamiast istniejącej nazwy tabeli wykorzystany zostaje alias, więc wcześniej trzeba wydobyć z zapytania wszystkie aliasowania. Trzeci przypadek jest również trudny, ponieważ kolumna należy do jednej z wykorzystanych w zapytaniu tabel, więc trzeba wiedzieć dodatkowo, jakie tabele są dostępne. Dla tych dwóch przypadków sytuację dodatkowo komplikuje fakt, że zapytania mogą być swobodnie zagnieżdżanie i łączone szeregowo za pomocą operatorów zbiorowych, co skutkuje pojawieniem się w obrębie pojedynczych zapytań zakresów, w których dostępne są różne tabele i różne aliasowania. Prosta analiza zbioru `Spider` wykazała, że 15,6% zapytań takie zakresy zawiera.

Aby poradzić sobie z ustaleniem przynależności kolumn do tabel w skomplikowanych zapytaniach postanowiono dokonywać rekurencyjnego rozbijania składających się na nie tokenów na zapytania prostsze. Przekazywany jest przy tym kontekst mówiący o obowiązującym aliasowaniu z zapytań zewnętrznych do wewnętrznych. Zostało to zilustrowane na rysunkach 4.4 oraz 4.5. Przedstawiają odpowiednio dekompozycję zapytania zawierającego operator zbiorowy oraz dekompozycję zapytania z zagnieżdżeniem. W wyniku takiego rekurencyjnego procesu rozbijania otrzymywany jest zbiór elementarnych zapytań wraz z obowiązującymi wewnątrz nich kontekstami, co pozwala na łatwe przetłumaczenie tokenów będących nazwami tabel i kolumn. Jako, że pierwotnym źródłem tych tokenów jest wejściowe, skomplikowane zapytanie, to również ono jest tłumaczone – niejako jako efekt uboczny, lecz było to celem od samego początku.



Rys. 4.4: Metoda dekomponowania zapytań z operatorami zbiorowymi



Rys. 4.5: Metoda dekomponowania zapytań z zagnieżdżeniem. *Fragment w zapytaniu zewnętrznym odpowiadający zapytaniu wewnętrznemu zastępowany jest wyrażeniem* `SELECT 1`*, aby zapewnić strukturalną poprawność obu rezultatów dekompozycji.*

4.3.3 Rekonstrukcja informacji redundantnych

Jak zostało wcześniej wskazane, dane źródłowe, na których bazuje algorytm generacji, zawierają jedynie informacje niezbędne. W pełnym zbiorze muszą znaleźć się również te, które zostały uznane za redundantne i w danych źródłowych pominięte. Redundancja ta nie polega na dokładnych powtórzeniach, bo usunięte informacje stanowią pochodne innych. Dokonanie ich rekonstrukcji stanowi sedno niniejszej części.

Znaczną część koniecznych do odtworzenia informacji stanowią pytania i zapytania SQL podzielone na tokeny. Okazuje się jednak, że duża liczba istniejących rozwiązań w ogóle nie korzysta z tych informacji, ponieważ tokenizacji można dokonać na różne sposoby. Zamiast tego wykonują ją samodzielnie, wybierając sposób dla siebie najkorzystniejszy. Niemniej jednak pierwsze modele (najwcześniej opracowane), jak i zapewne część nowszych, korzysta z dostarczonego w ramach zbioru sposobu tokenizacji, więc należy o ten aspekt mimo wszystko zadbać.

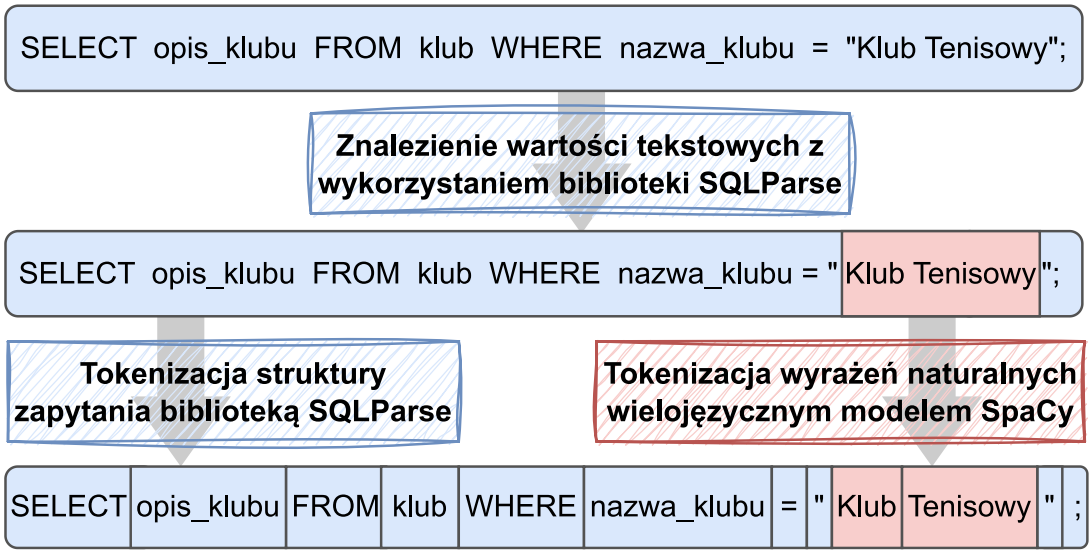
Tokenizacja pytań (atrybut question_toks)

Pytania podzielone na tokeny powinny znaleźć się w finalnych próbkach pod atrybutem o nazwie `question_toks`. Tokenizacji postanowiono dokonywać z wykorzystaniem dostarczonego w ramach biblioteki `SpaCy` wielojęzycznego modelu. Wybrano akurat wariant wielojęzyczny, a nie polski, ponieważ dzięki temu skrypt generujący nie będzie wprowadzał dodatkowych ograniczeń

co do języka pytań, a generowanie zbiorów, chociażby z angielskimi pytaniami, również okazało się przydatne. Porównano jednocześnie rezultaty tokenizacji polskich pytań za pomocą modelu wielojęzycznego oraz polskiego. Okazuje się, że wyniki różnią się jedynie w niewielkiej liczbie przypadków, co świadczy o tym, że model wielojęzyczny jest wysokiej jakości.

Tokenizacja zapytań SQL (atrybut query_toks)

Zapytania SQL podzielone na tokeny to element, który również musi znaleźć się w kompletnym zbiorze. Występuje w próbkach pod postacią atrybutu `query_toks`. Zaproponowany sposób tokenizacji opiera się na początkowym znalezieniu w zapytaniu wszystkich wartości tekstowych. Wartości te są następnie tokenizowane za pomocą wspomnianego wcześniej wielojęzycznego modelu z biblioteki `SpaCy`. Reszta zapytania jest natomiast rozbijana na elementy składowe z wykorzystaniem biblioteki `SQLParse`. Proces ten został zilustrowany na rysunku 4.6.



Rys. 4.6: Schemat działania zaproponowanego algorytmu tokenizacji zapytań SQL

Tokenizacja zapytań SQL bez wartości (atrybut query_toks_no_value)

Format zbioru `Spider` wymaga od próbek posiadania atrybutu `query_toks_no_value`, który musi zawierać zapytanie SQL podzielone na tokeny, lecz bez wartości. Mówiąc bardziej precyzyjnie, wartości powinny zostać zamaskowane za pomocą specjalnego tokena `value`. Łatwo zauważyć, że atrybut `query_toks_no_value` jest bardzo podobny do opisywanego chwilę wcześniej `query_toks`. Postanowiono więc wykorzystać identyczną technikę, lecz z jedną róż-

nicą: znalezione w zapytaniach wartości, zamiast podlegać rozbiciu na tokeny, są po prostu zastępowane wspomnianym tokenem specjalnym.

Opracowany algorytm jest niemal całkowicie zbieżny z wykorzystanym w oryginalnym zbiorze `Spider` sposobem tokenizacji zapytań. Aby to zweryfikować, dokonano za jego pomocą podziału oryginalnych zapytań i sprawdzono, czy wyniki pokrywają się z oryginalnymi tokenami. Okazuje się, że różnice występują jedynie w 18 przypadkach spośród prawie dziesięciu tysięcy. Dokładniejsza analiza tych kilkunastu przykładów sugeruje, że zapytania SQL w nich zawarte zostały podzielone na tokeny w niekonsekwentny, odbiegający od reszty sposób. Zbiór `Spider` nie jest więc do końca spójny. Brak konsekwencji to jeden z problemów, którym rozwijana strategia generowania zbiorów powinna przeciwdziałać i jak widać, ma to rzeczywiście znaczenie.

Parsowanie zapytań SQL (atrybut `sql`)

Ważnym elementem, który musi się znaleźć w finalnym zbiorze, są odpowiednio sparsowane zapytania SQL. Mają swoje miejsce w próbkach pod kluczem `sql`. Tym razem istnieje publicznie dostępny skrypt, który dokonuje tego procesu. Stanowi go plik `parse_sql.py`, znajdujący się w repozytorium kodu (T. Yu, Zhang, K. Yang i in. 2018b) dostarczonym przez twórców `Spider`. Zostały napotkane jednak pewne problemy wynikające z jego niedociągnięć. W jednym z pierwszych kroków przeprowadza on bowiem operację wydobycia z podanego zapytania aliasowania, czyli słownika mówiącego, jakie występują w nim aliasy i na jakie tabele się mapują. Niestety dokonuje tego globalnie, a jak zostało przedyskutowane wcześniej, nie sprawdza się to dla wszystkich zapytań, ponieważ występują w nich zakresy – w jednej części zapytania dany alias może mapować się na zupełnie inną tabelę niż w drugiej części. Podczas uruchamiana tego skryptu na zapytaniach ze zbioru `Spider`, co jest najczęstszym scenariuszem, nie zwraca żadnych błędów. Przypuszcza się, że kłopotliwe zapytania zostały ręcznie poprawione tak, aby stały się parsowalne. W przypadku gdy jednak zostaną wcześniej poddane tłumaczeniu nazw tabel i kolumn, to problem ten wychodzi na wierzch i parsowanie kończy się najczęściej dla kilku zapytań niepowodzeniem.

Nasuwające się rozwiązania powyższego problemu są dwa: można poprawić skrypt parsujący lub zmodyfikować plik mapowania nazw schematu tak, aby problem się nie ujawnił. Postanowiono wybrać drugą opcję, ponieważ nie wymaga wiele wysiłku. Przypuszcza się, że podobny był również sposób myślenia twórców zbioru `Spider` – zaimplementowali uproszczony skrypt parsu-

jący, ponieważ koszt implementacji pełnego przewyższa wysiłek wymagany do jednorazowego dostosowania niewielkiej liczby zapytań. Niemniej jednak zaobserwowane zachowanie zostało zgłoszone na platformie GitHub w celu udokumentowania i zwrócenia na nie uwagi[12].

4.4 Wykonywanie tłumaczenia

We wcześniejszej części pracy przedstawiony został format danych źródłowych, wykorzystywanych do procesu generacji zbiorów. Zawiera on miejsca, w których znajdują się polskie tłumaczenia pytań, zapytań, nazw tabel oraz kolumn. Kwestia tego w jaki sposób tłumaczenia te uzyskać została do tej pory w dużym stopniu przemilczana – zostanie to opisane w kolejnych sekcjach.

4.4.1 Wybór tłumacza

Ważną do podjęcia decyzją, bo rzutującą w istotnym stopniu na jakoś finalnego zbioru danych, jest wybór narzędzia mającego być wykorzystanym do tłumaczenia maszynowego. Obecnie niemal wszystkie tego typu rozwiązania bazują na uczeniu głębokim i dlatego określane są terminem NMT (ang. Neural Machine Translation). Często są zastrzeżone i dostęp do nich uzyskuje się za pomocą webowego API. Istnieją również narzędzia typu Open Source, które można uruchomić w całości na własnym urządzeniu. Przykładem jest oprogramowanie OpenNMT (Klein, Kim, Deng, Senellart i in. 2017; Klein, Kim, Deng, V. Nguyen i in. 2018; Klein, Hernandez i in. 2020), bazujący na nim Argos Translate (Argos Open Tech 2020), czy TranslateLocally (Bogoychev i in. 2021). Pozwalają uniknąć naliczania kosztów i dają możliwość pewnej modyfikacji. Oferują za to mniejszą dokładność i dlatego postanowiono nie podążać tą drogą.

Podczas podejmowania ostatecznej decyzji wybór ograniczono do popularnych i renomowanych rozwiązań, takich jak Google Cloud Translation API (Google 2024), Microsoft Azure AI Translator (Microsoft 2024), Amazon Translate API (Amazon 2024) oraz DeepL (DeepL 2024). W szczególności to ostatnie wydaje się aktualnie wychodzić na prowadzenie pod względem jakości produkowanych tłumaczeń. Potwierdzają to liczne artykuły naukowe (Yulianto i in. 2021; Hidalgo-Ternero 2021; Bahasa i in. 2023), które powstały na fali zachwytu tym rozwiązaniem. DeepL posiada nawet bibliotekę do języka Python, znacznie ułatwiającą wykorzystanie.

[12] https://github.com/taoyds/test-suite-sql-eval/issues/19

Jedynym aspektem, który może zniechęcać do wykorzystania DeepL są związane z tym koszty, nieco wyższe niż w przypadku konkurencyjnych rozwiązań. W czasie tworzenia niniejszej pracy podstawowy plan opierał się na bazowej opłacie w kwocie 20 złotych na miesiąc oraz dodatkowym obciążeniu wynoszącym prawie 90 złotych za każdy przetłumaczony milion znaków. Dostępny jest jednak również darmowy plan, pozwalający na przetłumaczenie pół miliona znaków każdego miesiąca za darmo.

DeepL został ostatecznie wybranym narzędziem, a głównym tego powodem jest niekwestionowana jakość produkowanych przez to narzędzie tłumaczeń. Oszacowano, że dostępne w ramach darmowego planu limity powinny pozwolić na zrealizowanie postawionych celów. Nie uda się to jednak w jeden miesiąc, a konieczne będzie rozłożenie tłumaczeń na dłuższy okres.

4.4.2 Tłumaczenie nazw tabel i kolumn

Okazuje się, że wysokiej jakości tłumaczenie nazw tabel i kolumn jest wyjątkowo trudnym zadaniem i z tego powodu zostało ono podzielone na dwa etapy. Pierwszy zakłada wykorzystanie tłumacza maszynowego, a drugi ręczne poprawienie tłumaczeń. To ostatnie nie polega jednak na przeglądaniu każdego przykładu jeden po drugim i szczegółowej walidacji, lecz na bardziej holistycznym i heurystycznym podejściu, które zostanie dokładnie opisane.

Tłumaczenie maszynowe

Jak zaznaczono wcześniej, w zbiorze Spider dla tabel i kolumn występują podwójne nazwy: oryginalnie występujące w bazie danych (`first_name`) oraz w czytelnej, naturalnej postaci (`first name`). O ile przetłumaczenie tych ostatnich nie stanowi większego problemu, to nazwy oryginalne wymagają podjęcia dodatkowych działań. DeepL nie poradzi sobie z nimi w takiej postaci i jako tłumaczenie zwróci te same nazwy, bez dokonywania żadnych zmian (`first_name`). Wydaje się, że traktuje je jako niepodlegające tłumaczeniu identyfikatory. Zauważono, że Google Cloud Translation wykazuje pod tym kątem bardziej pożądane zachowanie, ale wciąż nie można by na nim polegać.

Aby poradzić sobie z zarysowanym problemem postanowiono przed tłumaczeniem dokonywać w nazwach podmiany znaków podkreślenia na spacje, by przekonwertować je do postaci naturalnej, a po tłumaczeniu spacje z powrotem zamieniać na znaki podkreślenia, co przedstawione zostało na rysunku 4.7. Mechanizm ten nie sprawdził się dla wszystkich przypadków,

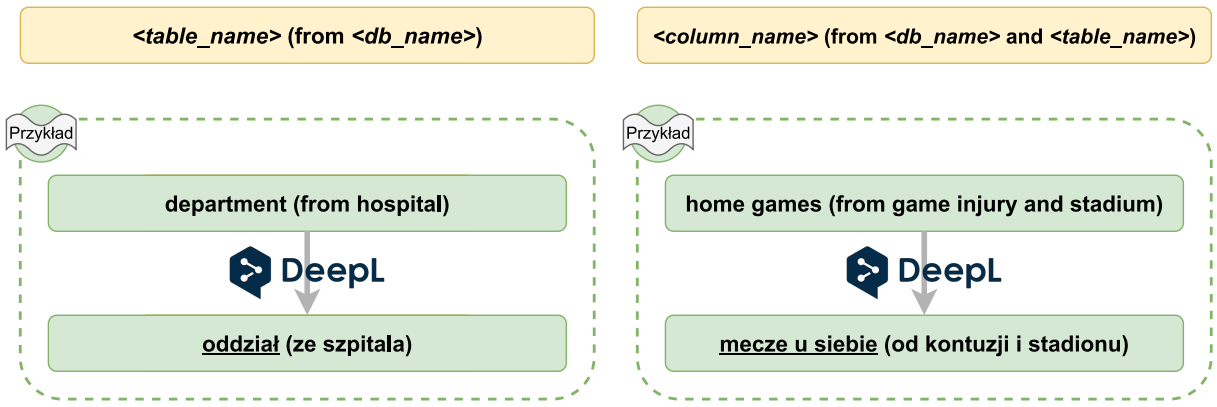
ponieważ niektóre wielowyrazowe nazwy były zapisywane za pomocą innych konwencji niż oddzielanie słów znakiem podkreślenia. Były one jednak w mniejszości i tymi niedociągnięciami postanowiono się zająć na etapie ręcznych poprawek.

Rys. 4.7: Schemat algorytmu do tłumaczenia oryginalnych nazw tabel i kolumn

Podstawowa metoda tłumaczenia nazw to ich proste i intuicyjne przekazywanie do tłumacza. Jest to sposób, który został prawdopodobnie wykorzystany we wszystkich tłumaczeniach maszynowych zbioru `Spider`, ponieważ ich autorzy nie zwrócili szczególnej uwagi na tę kwestię. Podczas realizacji niniejszej pracy zauważono jednak, że wiele błędów w takich tłumaczeniach wynika z braku uwzględniania przez tłumacza kontekstu i opracowano metodę, która pozwala w dużej mierze obejść ten problem. Dla przykładu kolumna `home_games` podstawową metodą jest tłumaczona jako `gry_domowe`, natomiast ulepszona metoda kontekstowa weźmie pod uwagę, że kolumna ta znajduje się w tabeli `stadium` oraz bazie danych `game_injury` i tym razem nazwa zostanie przetłumaczona poprawnie jako `mecze_u_siebie`.

Metoda kontekstowa polega na prostej obserwacji, że nowoczesne narzędzia, w szczególności `DeepL`, tłumaczą to samo słowo w różny sposób w zależności od kontekstu, w jakim ono wystąpiło. Z tego powodu do tłumacza poza samymi nazwami tabel i kolumn postanowiono także podawać w roli kontekstu nazwy baz danych, a dla kolumn również nazwy tabel. Zostały opracowane szablony, które służą do tworzenia skontekstualizowanych wyrażeń podawanych do tłumacza i wraz z przykładami użycia zostały przedstawione na rysunku 4.8. Dla porównania postanowiono dokonać tłumaczenia bezkontekstowego i okazało się, że wynikowe tłumaczenia nazw pomiędzy tymi metodami różnią się dla 21,41% przypadków, co stanowi dużą różnicę. W tabeli 4.1 przedstawiono zestawienie kilku nazw przetłumaczonych kontekstowo oraz bezkontekstowo.

Przedstawiona strategia tłumaczenia kontekstowego jest zainspirowana metodą `SAVe`, opisaną w artykule dotyczącym wielojęzycznego zbioru `MultiSpider` (Dou i in. 2023). Została tam wykorzystana jako metoda augmentacji dla poprawienia osiąganych przez model wyników. Jest bardziej skomplikowana od przedstawionego powyżej wariantu, ponieważ zakłada wykorzystanie wielu szablonów oraz ma dodatkowy etap weryfikacji tłumaczeń, co jednak dla rozważanego problemu nie ma dobrego uzasadnienia.



(a) Dla tabel **(b)** Dla kolumn

Rys. 4.8: Szablony do tłumaczenia kontekstowego

Słowo tłumaczone	Nazwa bazy	Czy kolumna	Nazwa tabeli	Tłumaczenie bezkontekstowe	Tłumaczenie kontekstowe
major	club	tak	student	główny	kierunek
sex	new pets	tak	student	seks	płeć
return data	sakila	tak	rental	data powrotu	data zwrotu
players	wta	nie	—	gracze	zawodnicy
available policies	insurance fnol	nie	—	dostępne zasady	dostępne polisy
likes	network	nie	—	upodobania	polubienia

Tabela 4.1: Porównanie kontekstowego i bezkontekstowego tłumaczenia nazw

Oczywistym minusem zaproponowanej metody jest istotne zwiększenie długości tekstów podawanych do tłumacza, co ze względu na koszty należy ograniczać. Zaletą jest poprawa tłumaczenia dla wielu nazw, jednak z pewnego punktu widzenia może ona także wpływać na obniżenie czytelności przetłumaczonego schematu. Zdarzają się bowiem przypadki, iż w obrębie jednej bazy danych pewna nazwa kolumny zostaje przetłumaczona na kilka różnych sposobów, chociaż nie powinna. Zmniejsza to spójność tak przetłumaczonego schematu, bo sugeruje, że przechowywane są w tych kolumnach dane o różnym znaczeniu, a w rzeczywistości jest inaczej. Nie jest więc banalnym oszacowanie wypadkowego wpływu zastosowania tej metody na jakość finalnego zbioru, uważa się jednak, że jest on pozytywny.

Ręczne poprawki

Po uzyskaniu tłumaczeń maszynowych dla nazw tabel i kolumn oraz ich wstępnej ocenie stało się oczywiste, że wciąż zawierają pewne błędy. Przejrzenie każdego z nich jedno po drugim wymagałoby znacznego nakładu czasu, więc postanowiono posłużyć się metodami heurystycznymi, by zlokalizować tylko te najbardziej podejrzane tłumaczenia.

Zauważono, że dla błędnych tłumaczeń bardzo często nazwa przetłumaczona jest identyczna z angielską, więc korzystając z tego faktu, prostych skryptów oraz możliwości edytora `Visual Studio Code` dokonano znalezienia właśnie takich przypadków, a następnie je poprawiono. Były to w dużej mierze nazwy składające się z kilku wyrazów połączonych ze sobą za pomocą innej konwencji niż znaki podkreślenia, bo tylko ten najpopularniejszy wariant był obsługiwany w fazie maszynowej. Dość częste były również kilkuliterowe skróty jak `mid` (movie id), czy `aid` (author id), z którymi – co nie dziwi – `DeepL` sobie nie poradził. Poza tym dokonano poprawy kilkunastu wyrażeń, w przypadku których zaobserwowano częste pomyłki, jak tłumaczenie `name` na `imię i nazwisko`, chociaż powinno zostać przetłumaczone jako `nazwa`, czy `id`, które niepotrzebnie zostało rozwinięte na rozwlekłą nazwę `identyfikator`.

Po dokonaniu poprawek związanych z jakością tłumaczeń należało zweryfikować je pod kątem konfliktów nazw. Konieczne było upewnienie się, że w obrębie jednej bazy nie powstało kilka tabel o takich samych nazwach, czy też kilka identycznych kolumn w obrębie tabeli. W tym celu napisano skrypt w języku Python, który analizuje podane mapowanie nazw schematu i raportuje znalezione problemy, które następnie ręcznie naprawiono. Takich konfliktów było zaledwie kilka, lecz ich rozwiązanie okazało się zaskakująco trudne. Dwa przykłady to konflikt nazw kolumn `result` i `score`, które zostały przetłumaczone jako `wynik` oraz tabel `Type_Of_Restaurant` i `Restaurant_Type`, które zostały przetłumaczone jako `Typ_Restauracji`. W tym drugim przypadku szczególnie trudno było znaleźć dwa różne polskie tłumaczenia i postanowiono zastosować nie do końca satysfakcjonującą parę `Typ_Restauracji` oraz `Restauracja_Typu`.

Na cały etap ręcznych poprawek zostało poświęcone kilka godzin i w jego wyniku zmodyfikowano 10,54% tłumaczeń maszynowych. Wydaje się to dużą częścią, lecz większość z tych modyfikacji dokonano w szybki, półautomatyczny sposób.

4.4.3 Tłumaczenie pytań

Tłumaczenie pytań sprowadza się do uzupełnienia w przedstawionym formacie danych źródłowych atrybutu `question.pl` na podstawie `question.en`. W tym przypadku pytania zostały bezpośrednio przekazane do tłumacza, bez stosowania żadnych dodatkowych sztuczek. Uznano, że stosowanie tłumaczenia kontekstowego, które oznaczałoby rozszerzenie przekazywanych do tłumacza tekstów o nazwę bazy danych z której pochodzą, nie ma sensu. Wiązałoby się to z tłumaczeniem większej liczby znaków na podstawie których `DeepL` obciąża swoich użytkowników,

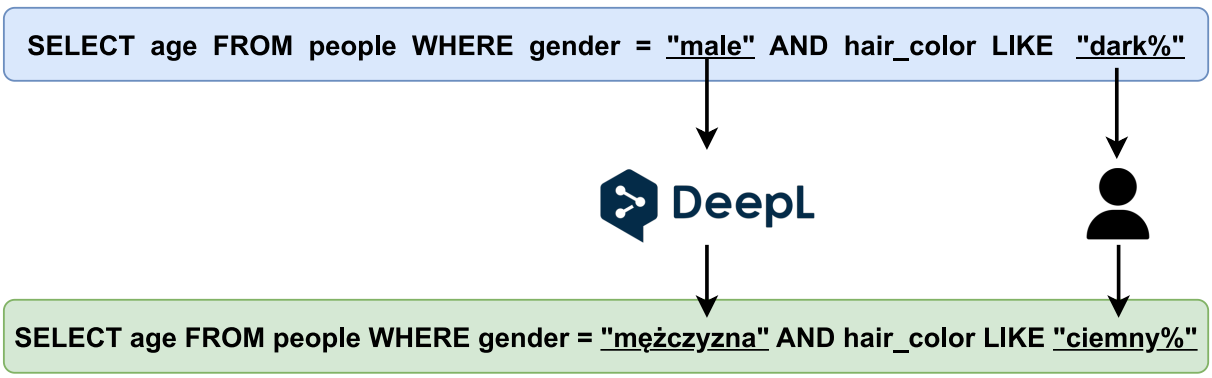
a uzyskana poprawa prawdopodobnie nie byłaby znacząca. Pytania są bowiem zwykle na tyle długie, że pozwalają tłumaczowi na wywnioskowanie domeny, której dotyczą, więc wybranie odpowiednich tłumaczeń dla problematycznych słów nie stanowi już tak dużego wyzwania.

Etap manualnych poprawek nie został w tym przypadku wykorzystany, gdyż pytania zawierają dużą ilość tekstu i trudno jest dostrzec w nich ewentualne błędy. Nie znaleziono również żadnych sposobów na odszukanie podejrzanych tłumaczeń, czy dokonanie półautomatycznych poprawek, tak jak to miało miejsce dla tłumaczenia nazw tabel i kolumn. Ostatecznie maszynowe tłumaczenia pytań wydawały się na tyle wysokiej jakości, że postanowiono przejść do kolejnych etapów.

4.4.4 Tłumaczenie wartości w zapytaniach SQL

Tłumaczenie wartości w zapytaniach SQL odbywa się poprzez uzupełnienie atrybutu `query.pl` na podstawie `query.en` w pliku danych źródłowych. Ten ostatni atrybut zawiera oryginalne zapytania SQL, natomiast pierwszy zapytania z wszelkimi anglojęzycznymi łańcuchami znaków przetłumaczonymi na język polski.

W celu znalezienia w danym zapytaniu wartości tekstowych dokonywano jego przetwarzania z wykorzystaniem biblioteki `sqlparse`. Pozwalała ona podzielić zapytanie na poszczególne tokeny i wybrać te, które posiadają typ `Token.Literal.String`, czyli stanowią wartości tekstowe. Są to wyrażenia zapisane się w cudzysłowach, które przekazywano do `DeepL` w celu uzyskania tłumaczeń, a następnie zastępowano nimi oryginalne teksty. Wyjątek stanowią wartości występujące w roli szablonów po słowie kluczowym `LIKE`, takie jak `'%Hey%'`, `'%East%'`, `'S%'`. Nie są zapisane w języku naturalnym, ponieważ zawierają specjalne znaki, stanowiące problem dla tłumacza maszynowego. Biorąc pod uwagę rzadkość takich przypadków, postanowiono przetłumaczyć je manualnie. Wizualizacja tego etapu została przedstawiona na rysunku 4.9.



Rys. 4.9: Wizualizacja etapu tłumaczenia wartości w zapytaniach SQL

4.5 Stworzone zbiory

Niniejszy część stanowi ukoronowanie pracy nad polskimi zbiorami danych, ponieważ zawiera ich ostateczne przedstawienie wraz z analizą. Zgodnie z obraną strategią kompletne zbiory stanowią połączenie jednego z bazowych zbiorów oraz jednego z mapowań nazw tabel i kolumn, dlatego komponenty te zostaną opisane osobno. Na koniec kilka ich połączeń zostanie nazwanych w celu ułatwienia dalszych rozważań.

4.5.1 Zbiory bazowe

Przygotowanych zostało pięć bazowych zbiorów, z których zdecydowanie najistotniejszym ze względu na swoją renomę oraz powszechne wykorzystanie jest `Spider`. W celu weryfikacji i porównywania trenowanych modeli przetłumaczone zostały także zbiory `Spider-DK` oraz `Spider-Syn`. `CoSQL` oraz `SParC` to natomiast znacznie różniące się od swoich oryginałów zbiory, które przygotowano i przetłumaczono głównie w celu uzyskania dodatkowych danych treningowych.

Zbiór	Część treningowa	Część walidacyjna	Razem
Spider	8659	1034	9693
Spider-Syn	3556	773	4329
Spider-DK	0	535	535
CoSQL	1562	230	1792
SParC	2651	366	3017
Razem	16428	2938	19366

Tabela 4.2: Zestawienie liczby próbek w poszczególnych zbiorach

W tabeli 4.2 przedstawione zostały liczności próbek we wszystkich stworzonych zbiorach bazowych z uwzględnieniem podziału na części treningowe oraz walidacyjne. `Spider` to zbiór najliczniejszy, z którego żadne próbki nie były usuwane i dokładnie odpowiadają oryginalnym. Drugim największym jest `Spider-Syn`, który zawiera przykłady ze `Spider` z niewielkimi modyfikacjami, lecz w wyniku usuwania duplikatów skurczył się o ponad połowę. Zbiory `SParC` oraz `CoSQL` są następne w kolejności i one również w czasie deduplikacji zostały uszczuplone. Najmniejszym zbiorem, posiadającym jedynie część walidacyjną, jest `Spider-DK`, chociaż z niego żadne próbki usuwane nie były.

Repozytorium zbioru `Spider` (T. Yu, Zhang, K. Yang i in. 2018b) dostarcza algorytm pozwalający przypisać do każdego zapytania SQL jeden z czterech poziomów trudności: `Easy`, `Medium`,

Zbiór	Easy	Medium	Hard	Extra Hard
Spider	2231 (23%)	3445 (36%)	2095 (22%)	1922 (20%)
Spider-DK	110 (21%)	246 (46%)	74 (14%)	105 (20%)
Spider-Syn	952 (22%)	1825 (42%)	884 (20%)	668 (15%)
CoSQL	949 (53%)	488 (27%)	222 (12%)	133 (7%)
SParC	2131 (71%)	706 (23%)	138 (5%)	42 (1%)
Wszystkie	6373 (33%)	6710 (35%)	3413 (18%)	2870 (15%)

Tabela 4.3: Zestawienia liczby próbek o poszczególnych poziomach trudności

`Hard` oraz `Extra Hard`. Odbywa się to poprzez zliczenie w zapytaniu różnych komponentów i klasyfikację na podstawie zdefiniowanych reguł. Przykłady zapytań o poszczególnych poziomach trudności przedstawiono na rysunku 4.10. Postanowiono dokonać analizy rozłożenia zapytań o poszczególnych poziomach trudności we wszystkich zbiorach. Wyniki tej analizy umieszczono w tabeli 4.3. Można je podsumować twierdzeniem, że najbardziej skomplikowane zapytania występują w zbiorze `Spider`, co niewątpliwie jest istotnym powodem jego popularności. Niemal równie trudne zapytania posiada zbiór `Spider-DK`, lecz jest on wielokrotnie mniejszy. Z drugiej strony wyjątkowo proste okazały się zapytania znajdujące w oryginalnie kontekstowych zbiorach `CoSQL` oraz `SParC`.

Zbiór	Pytania (en / pl)	Wartości (en / pl)
Spider	66,74 / 66,44	10,66 / 11,28
Spider-DK	66,01 / 65,93	8,47 / 8,59
Spider-Syn	74,03 / 74,14	9,65 / 10,28
CoSQL	53,05 / 52,66	10,11 / 10,72
SParC	44,94 / 45,52	10,24 / 10,84
Wszystkie	63,69 / 63,61	10,32 / 10,94

Tabela 4.4: Zestawienie średniej liczby znaków w pytaniach i wartościach

Interesującą analizą jest zestawienie długości tekstów angielskich oraz przetłumaczonych na język polski. Zgodnie z danymi ukazanymi w tabeli 4.4 średnie długości wszystkich pytań liczone w znakach różnią się zaledwie o 0,08. W przypadku długości wartości tekstowych różnica ta jest nieco większa i wynosi 0,61. Wzrost może być spowodowany faktem, że występują one jedynie w niektórych próbkach i ze statystycznego punktu widzenia liczba ta jest mniej pewna. Mimo wszystko różnice okazały się zdecydowanie mniejsze od spodziewanych. Oznacza to, że długość pytań i wartości nie będzie miała wpływu na utrudnienie, czy też ułatwienia tłumaczenia zapytań w stosunku do języka angielskiego.

```sql
SELECT COUNT(*)
FROM cars_data
WHERE cylinders > 4
```

(a) Easy

```sql
SELECT T2.name, COUNT(*)
FROM concert AS T1
JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
GROUP BY T1.stadium_id
```

(b) Medium

```sql
SELECT T1.country_name
FROM countries AS T1
JOIN continents AS T2
ON T1.continent = T2.cont_id
JOIN car_makers AS T3
ON T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

(c) Hard

```sql
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
  (SELECT T1.name
   FROM country AS T1
   JOIN country_language AS T2
   ON T1.code = T2.country_code
   WHERE T2.language = "English"
    AND T2.is_official = "T")
```

(d) Extra Hard

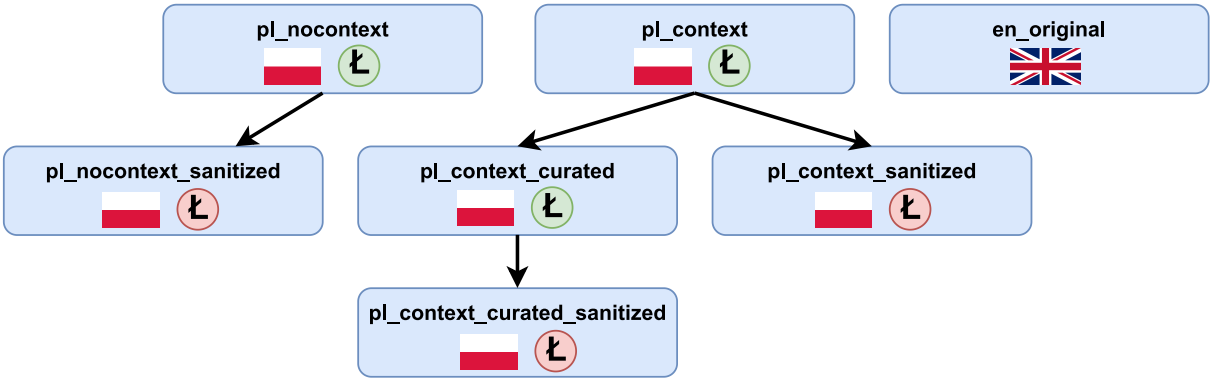
Rys. 4.10: Przykłady zapytań SQL o różnych poziomach trudności

4.5.2 Mapowania nazw

Zgodnie z przedstawionymi wcześniej informacjami opracowane zostały dwie główne drogi dokonywania tłumaczeń nazw tabel i kolumn: metoda bezkontekstowa oraz uważana za lepszą metoda kontekstowa. Mapowania nazw uzyskane z wykorzystaniem tych strategii nazwano odpowiednio `pl_nocontext` oraz `pl_context`. W tłumaczeniach maszynowych powstałych metoda kontekstową dokonano następnie dużej liczby półautomatycznych poprawek i stworzone w ten sposób mapowanie nazwano `pl_context_curated`. Przewidziano również mapowanie zachowujące oryginalne, angielskie nazwy, które określono jako `en_original`.

Wymienione mapowania zawierają polskie znaki, które jednak rzadko są spotykane w praktyce, więc opracowano skrypt, który dokonuje ich zastąpienia występującymi w kodowaniu `ASCII` odpowiednikami, jak `wysokość -> wysokosc`. W ten sposób powstały odpowiadające trzem wcześniej wspomnianym mapowania z sufiksem `_sanitized`. Schemat przedstawiający naszkicowane zależności został umieszczony na rysunku 4.11.

Podobnie jak to było w przypadku pytań, tutaj również postanowiono zbadać, w jaki sposób zastosowanie poszczególnych mapowań wpływa na długość finalnych nazw tabel i kolumn. Wyniki tej analizy przedstawiono w tabeli 4.5. Można zauważyć, że tłumaczenie na język polski zwiększa długości o około jeden znak, co stanowi prawie 10%. Jest to znaczny wzrost, biorąc pod



Rys. 4.11: Schemat zależności pomiędzy stworzonymi mapowaniami nazw. *Litera Ł w zielonym kółku oznacza obecność polskich znaków, natomiast w czerwonym ich brak.*

uwagę, że w przypadku tłumaczenia pytań był on marginalny. Wyjaśniać może to obserwacja, że angielskie nazwy tabel i kolumn często bywają zapisywane skrótowo, a tłumaczenia produkowane przez `DeepL` są bardziej opisowe. Przykładowo nazwa tabeli `Student_Tests_Taken` została przetłumaczona jako `Testy_Wykonane_Przez_Uczniów`.

Nazwy mapowań	Tabele		Kolumny	
	Nazwy naturalne	Nazwy oryginalne	Nazwy naturalne	Nazwy oryginalne
`en_original`	10,48	10,28	10,26	9,87
`pl_nocontext`	11,37	11,17	11,24	10,83
`pl_context`	11,46	11,23	11,38	11,11
`pl_context_curated`	11,55	11,29	11,36	11,07

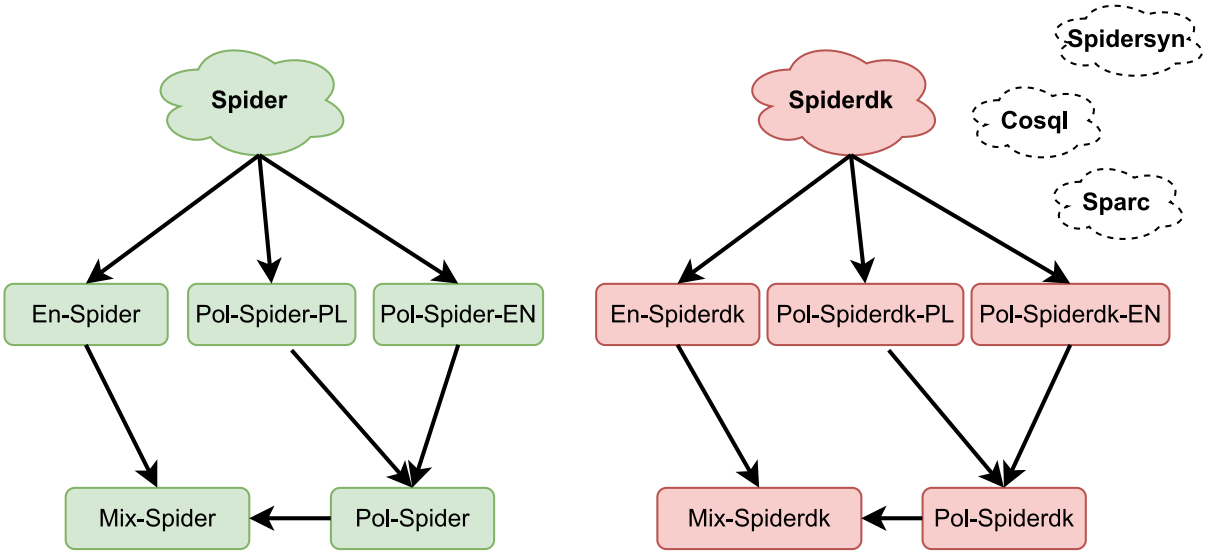
Tabela 4.5: Zestawienie średniej liczby znaków w mapowaniach nazw

4.5.3 Nazwane warianty

W celu ułatwienia dalszej pracy poprzez możliwość odwoływania się do konkretnych zbiorów kilka ich konfiguracji zostało nazwanych. Ponadto zdefiniowane zostały pewne połączenia tych zbiorów, ponieważ występują do nich częste odwołania.

Schemat zdefiniowanych zbiorów został przedstawiony na rysunku 4.12. Można z niego odczytać, że na bazie zbioru `Spider` wygenerowane zostały trzy zbiory pochodne. `En-Spider` to wariant całkowicie angielski, bardzo podobny do oryginalnego `Spider`, ale z minimalnymi różnicami, jak sposób tokenizacji. Wykorzystuje angielskie pytania i wartości oraz mapowanie `en_original`. Kolejny to `Pol-Spider-PL`, czyli wariant polski, z polskim schematem baz danych, uzyskanym za pomocą mapowania `pl_context_curated_sanitized`. Trzecim jest

natomiast `Pol-Spider-EN`, czyli również zbiór polski, ale z angielskim schematem baz danych, stworzonym za pomocą mapowania `en_original`.



Rys. 4.12: Schemat zależności pomiędzy zbudowanymi zbiorami danych

Opracowany został dodatkowo skrypt, który pozwala na łatwe łączenie zbiorów. Za jego pomocą dokonano połączenia dwóch wspomnianych polskich wariantów w zbiór, który nazwano `Pol-Spider`. Ma on podwójny rozmiar i obejmuje dwojaki sposób nazewnictwa kolumn i tabel. Jest istotny z praktycznego punktu widzenia, ponieważ nie ogranicza się tylko do pierwszego, czy drugiego przypadku, lecz obejmuje obie powszechnie spotykane konwencje. Po rozszerzeniu tego już rozbudowanego polskiego zbioru o angielski `En-Spider` powstaje natomiast `Mix-Spider`, czyli dość nietypowy zbiór zawierający próbki w obu językach.

Podobny schemat nazewnictwa do przedstawionego powyżej został zastosowany do reszty zbiorów bazowych, czyli `Spider-DK`, `Spider-Syn`, `CoSQL` oraz `SParC`. Wykorzystano wymowne i schematyczne nazwy, aby ułatwić ich zapamiętanie. Możliwych do wygenerowania konfiguracji zbiorów jest znacznie więcej, lecz do tych przedstawionych powyżej będą występowały częstsze odwołania w dalszej części pracy.

5 Przegląd podejść do problemu Text-to-SQL

Celem niniejszego rozdziału jest przedstawienie zwięzłego przeglądu podejść do problemu Text-to-SQL. Najpierw wskazane zostaną popularne metryki, wykorzystywane do pomiaru skuteczności tworzonych algorytmów, a następnie naszkicowane wysokopoziomowe podejścia do tego problemu, które ukształtowały się na przestrzeni lat. Zawarta wiedza stanowi podstawę do zrozumienia kolejnej części, w ramach której analizowane i testowane będą różne modele uczenia maszynowego. W celu głębszego zapoznania się z poruszanymi kwestiami można sięgnąć do jednego z wielu artykułów dokonujących przeglądu problemu tłumaczenia zapytań (Katsogiannis-Meimarakis i in. 2021; Quamar i in. 2022; Katsogiannis-Meimarakis i in. 2023).

5.1 Stosowane metryki

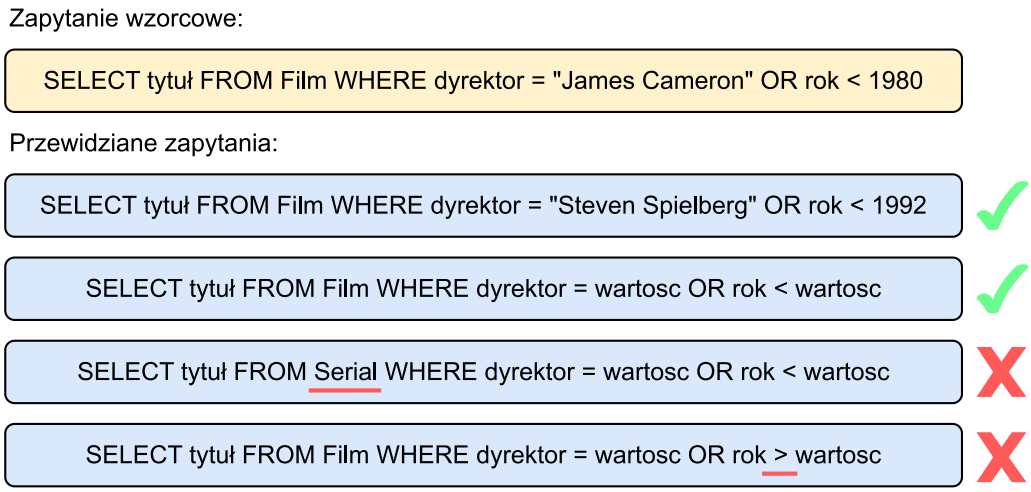
Posiadając docelowe zapytanie, które w literaturze anglojęzycznej określane jest mianem gold query, oraz zapytanie wyprodukowane przez dowolny algorytm, zachodzi potrzeba dokonania oceny tego, w jakim stopniu one sobie odpowiadają. Pozwala to bowiem w liczbowy sposób wyrazić dokładność algorytmu i dokonać jego porównań z innymi. Manualna ocena zwykle nie wchodzi w grę ze względu na swoją czasochłonność, więc opracowanych zostało kilka metryk, które dokonują tego w sposób automatyczny. Te najważniejsze opisano poniżej.

5.1.1 Exact Set Match (EM)

Problemem zwyczajnego porównywania ze sobą docelowych i przewidywanych zapytań SQL jako wartości tekstowych jest to, że mogą się różnić w kolejności elementów niemających znaczenia. Przykładowo zwykle nieistotne jest, w jakiej kolejności zostaną zwrócone poszczególne kolumny. Podobnie nie ma znaczenia uporządkowanie wyrażeń boolowskich połączonych operatorami AND, czy OR. Ostatecznie obojętna jest nawet kolejność występowania klauzuli SELECT, FROM, czy WHERE w strukturze zapytania. Metryka Exact Set Match, nazywana skrótowo EM, rozwiązuje naszkicowany problem poprzez odpowiednie przetworzenie zapytań i porównywanie poszczególnych składników, traktując je jak zbiory, czyli ignorując kolejność, która jest nieistotna. Należy jednak zauważyć, że w języku SQL można osiągnąć ten sam cel przy pomocy kompletnie różnych instrukcji, zupełnie odmiennych strukturalnie. Rozważana metryka nie jest w stanie poprawnie rozpoznać takich przypadków i w rzeczywistości prawidłowe zapytanie oceni jako błędne.

5.1.2 Exact Set Match Without Values (EM Without Values)

Okazuje się, że szczególnie trudnym do przewidywania fragmentem zapytań SQL są wartości, czyli występujące w nich łańcuchy tekstowe i liczby. Powstające kiedyś modele bardzo słabo radziły sobie z nimi, a często w ogóle ich nie przewidywały – produkowały zapytania, w których wartości były zamaskowane. Sprawiało to, że zwracane instrukcje nie były do końca prawidłowe, czy kompletne, lecz jeżeli reszta komponentów była odpowiednia, to chciano całe zapytanie zaklasyfikować jako poprawne. W tym celu opracowana została metryka `Exact Set Match Without Values`. Jej mechanizm działania jest niemal identyczny jak `Exact Set Match`, bo jedyną różnicą jest ignorowanie rozbieżności w wartościach. Przykłady kilku poprawnych i niepoprawnych zapytań dla metryki `EM Without Values` przedstawiono na rysunku 5.1.



Rys. 5.1: Prawidłowe i nieprawidłowe zapytania dla metryki `EM Without Values`

5.1.3 Execution Accuracy (EX)

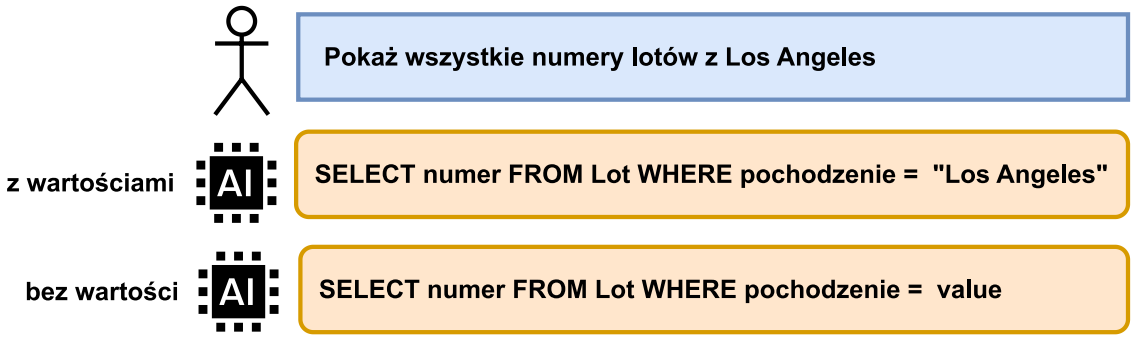
Metryka `Execution Accuracy`, nazywana skrótowo `EX`, wychodzi z prostego założenia, że jeżeli przewidziane zapytanie jest poprawne, to jego wykonanie musi zwrócić takie same rekordy jak zapytanie referencyjne. Ma ona jednak pewne wymagania. Przede wszystkim porównywane zapytania muszą być wykonywalne, więc zawierać wartości. Drugim warunkiem jest posiadanie wypełnionych baz danych, ponieważ w przeciwnym wypadku wszystkie zapytania będą zwracały pustą listę rekordów i tym samym zostawały błędnie uznane za poprawne. Posiadając nawet duże bazy danych, mogą zdarzyć się przypadki, iż takie same rekordy zostaną przypadkowo zwrócone dla dwóch nierównoważnych zapytań, co skończy się błędną klasyfikacją. Stanowi to podstawowe ograniczenie metryki `EX`.

5.2 Kluczowe cechy modeli

Dość wyraźny jest podział istniejących algorytmów rozwiązujących problem `Text-to-SQL` ze względu na dwie cechy. Pierwsza z nich to kwestia obecności wartości w przewidywanych zapytaniach. Druga natomiast to posiadanie umiejętności wykorzystania zawartości baz danych. Informacje te w oficjalnym rankingu zbioru `Spider` są wyszczególnione dla każdego modelu, ponieważ mają dość istotne implikacje, które zostaną opisane w poniższych sekcjach.

5.2.1 Przewidywanie wartości

Algorytmy tłumaczące zapytania naturalne na język SQL dzielą się na dwie podstawowe kategorie: przewidujące kompletne zapytania oraz zapytania bez wartości. Wówczas produkują instrukcje zawierające w ich miejscuspecjalne symbole zastępcze, takie jak `value`, czy `terminal`. Przykłady wygenerowanych zapytań zgodnie z oboma podejściami przedstawiono na rysunku 5.2.



Rys. 5.2: Przykład wygenerowanych zapytań SQL z wartościami i bez

Przyczyną wyklarowania się takiego podziału jest fakt, iż wartości stanowią jeden z najtrudniejszych do wygenerowania fragmentów zapytań, a pierwsze z powstających modeli posiadały bardzo niską skuteczność. Słabo generowały wartości lub nie robiły tego wcale. Postanowiono więc obniżyć poziom postawionego sobie zadania poprzez zignorowanie wartości i przewidywanie samej struktury zapytań. Obecnie algorytmy posunęły się jednak znacząco do przodu i osiągają wysokie wyniki nawet w generowaniu kompletnych instrukcji SQL, więc uwaga badaczy przesunęła się na ten bardziej praktyczny przypadek.

5.2.2 Wykorzystanie zawartości baz danych

Drugą istotną cechą, na podstawie której można scharakteryzować wszystkie algorytmy tłumaczenia zapytań naturalnych na SQL, jest to, czy korzystają z zawartości baz danych. Podstawowym elementem, na którym operują, są bowiem schematy. Zawartość natomiast, pomimo że niejednokrotnie zawiera istotne dla generowanego zapytania informacje, to wymaga zwiększenia poziomu komplikacji algorytmu. Wiąże się to także z problemem wydajnego odpytywania nieraz ogromnych zbiorów. Mimo wszystko duża część istniejących modeli w jakiś sposób z zawartości baz danych korzysta.

5.3 Wysokopoziomowe podejścia

Na obecną chwilę w rankingu zbioru `Spider` znajdują się modele reprezentujące dwa wysokopoziomowe, całkowicie różne od siebie podejścia. Jedno z nich polega na projektowaniu i trenowaniu modeli dedykowanych, a drugie na wykorzystaniu dużych, pretrenowanych modeli językowych, gdzie żaden trening nie jest potrzebny.

5.3.1 Modele dedykowane

Przez modele dedykowane rozumie się takie, które zostały stworzone w celu rozwiązania tylko jednego, konkretnego problemu. Przykładem jest tłumaczenie pytań z języka naturalnego na SQL. Takie ograniczenie pozwala na uwzględnienie i uwypuklenie w architekturze modelu oraz procesie treningu wiele istotnej dla danej domeny wiedzy. Z tego powodu możliwe jest zmniejszenie liczby potrzebnych danych i skrócenie czasu treningu. Modele tego typu dla problemu `Text-to-SQL` istnieją od bardzo dawna i z biegiem czasu stają się coraz lepsze. W dalszej części rozdziału, w sekcji 5.4, zostało dokładniej opisane ich działanie. Jedynie w ostatnim czasie pojawiła się alternatywa w postaci dużych pretrenowanych modeli językowych.

5.3.2 Duże pretrenowane modele językowe

Duże pretrenowane modele językowe zostały zaadaptowane w bardzo wielu problemach. Są to sieci neuronowe wytrenowane na ogromnych zbiorach danych w celu zyskania wszechstronnej wiedzy. Zapoczątkowało je laboratorium badawcze `OpenAI`, które w roku 2018 opracowało pierwszy model językowy ogólnego przeznaczenia o nazwie `GPT-1` (Radford i in. 2018). Szczególną uwagę całego świata zwrócił na siebie jednak dopiero model `GPT-3.5`, który stanowi

kolejną iterację wcześniej wspomnianego. Najświeższym tego typu modelem i jednocześnie uważanym za najlepszy jest natomiast GPT-4-turbo, do którego badacze mogą uzyskać dostęp za pomocą przygotowanego API i zintegrować go ze swoimi rozwiązaniami.

Wykorzystanie dużych modeli językowych opiera się na traktowaniu ich jak czarnych skrzynek, które przyjmują na wejście instrukcję w języku naturalnym i zwracają odpowiedź. Te instrukcje wejściowe mają kluczowe znaczenie i określane są mianem promptów. Nawet niewielkie ich modyfikacji mogą drastycznie wpływać na otrzymywane odpowiedzi. W związku z tym powstała cała dyscyplina określana jako prompt engineering, która zajmuje się tworzeniem oraz optymalizacją promptów w takim kierunku, aby model zwracał oczekiwane odpowiedzi.

W ostatnim czasie rozwiązania bazujące na dużych, pretrenowanych modelach językowych zdominowały górną część rankingu zbioru Spider. Ich wadą jest jednak to, że wykorzystują płatne modele udostępniane przez OpenAI, co czyni je zbyt kosztownymi dla wielu zastosowań.

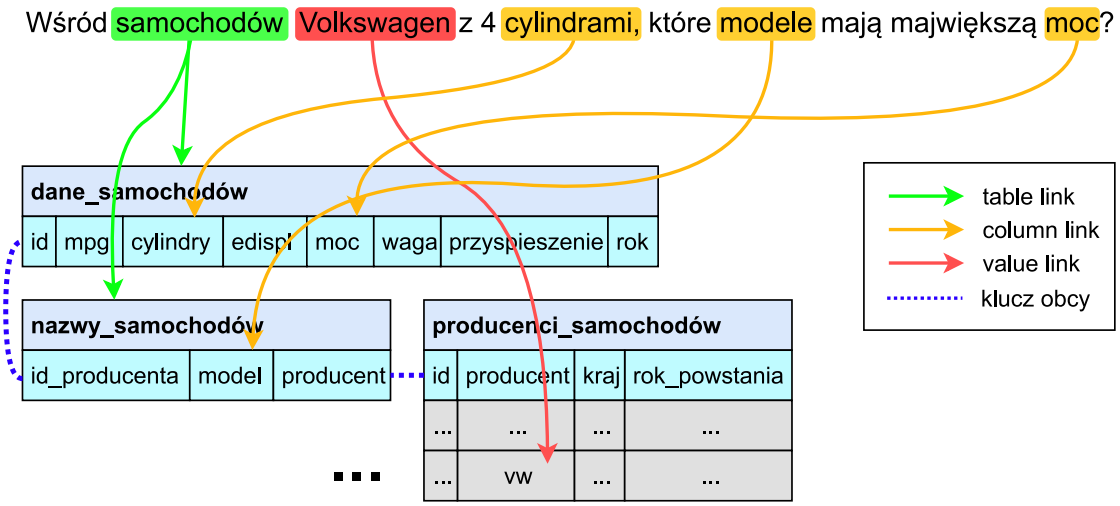
5.4 Etapy tłumaczenia zapytań

Celem niniejszej części jest przedstawienie powszechnie występujących w dedykowanych modelach Text-to-SQL etapów przetwarzania. Pozwoli to znacznie lepiej zrozumieć sposób działania rozwiązań wybranych do eksperymentów w kolejnym etapie.

Okazuje się, że niemal wszystkie powstałe do tej pory modele opierają się na architekturze enkoder-dekoder. Oznacza to, że występuje w nich etap enkodowania, podczas którego informacje wejściowe konwertowane są do zwięzłej reprezentacji pośredniej, a następnie w etapie dekodowania na jej podstawie generowana jest ostateczna odpowiedź. Etapy te mogą być zrealizowane na bardzo wiele sposobów. Zwykle wyróżnia się także dodatkową fazę określaną mianem schema linking, która zwraca informację pozwalającą lepiej dokonać procesu enkodowania.

5.4.1 Schema Linking

Celem etapu schema linking jest znalezienie powiązań pomiędzy frazami występującymi w pytaniu a elementami bazy danych, takimi jak tabele, kolumny i wartości. Jest to działanie, które jest intuicyjnie wykonywane nawet przez ludzkich ekspertów, kiedy stoją przed zadaniem skonstruowania nowego zapytania SQL. Przykładowy rezultat takiego działania przedstawiono na rysunku 5.3. W odróżnieniu od dwóch kolejnych etapów ten jest opcjonalny, lecz w praktyce często wykorzystywany.



Rys. 5.3: Przykład wykonania `schema linking`

`Schema linking` można rozbić na dwie fazy. Pierwsza polega na odnalezieniu wyrażeń w pytaniach oraz elementów w bazie danych, które mogą być w kontekście budowanego zapytania istotne. Nazywane są one odpowiednio `query candidates` oraz `database candidates`. Druga faza wymaga połączenia ze sobą tych elementów i w ten sposób skonstruowania połączeń nazywanych `table link`, `column link` lub `value link`, odpowiednio dla wystąpienia połączenia z tabelą, kolumną lub wartością.

Najprostsza metoda znajdywania kandydatów w pytaniu polega na uwzględnieniu każdego słowa z osobna. Wyrażenia stanowiące odwołania mogą składać się jednak z kilku słów, więc lepszym wyborem wydaje się rozważenie różnych n-gramów, czyli n-elementowych podciągów. Poza tym możliwe jest wykorzystanie analizy `NER` w celu znalezienia w pytaniach nazw własnych, które częstą okazują się tworzyć połączenia typu `value link`.

Oczywistymi kandydatami, jeśli chodzi o bazę danych, są wszystkie nazwy tabel i kolumn. Znacznie trudniej jest wybrać konkretne wartości, mogące tworzyć `value link`, bo z uwagi na ich wielość nie można brać wszystkich pod uwagę. Najczęściej dokonuje się jedynie poszukiwania w bazie konkretnych, podejrzanych wartości, które zostały ujawnione w pytaniu poprzez analizę `NER`, czy też zostały zapisane tam w cudzysłowie.

Po znalezieniu kandydatów w pytaniu i bazie danych należy spróbować dokonać ich dopasowania. Problem stanowi to, że pytania naturalne nie odwołują się najczęściej do elementów bazy poprzez dokładnie to samo określenie, lecz używają innych form i synonimów. Sprawia to, że dopasowywanie poprzez szukanie idealnych powtórzeń pomiędzy kandydatami często się nie sprawdza, lepszą metodą jest szukanie częściowych powtórzeń. Możliwe jest również

wykorzystanie odległości edycyjnej, czy odległości pomiędzy wektorowymi reprezentacjami wyrażeń.

5.4.2 Enkodowanie

Enkodowanie to etap, którego celem jest konwersja wszystkich danych wejściowych do zwartej, wektorowej postaci, czego dokonuje się za pomocą sieci neuronowej. Minimalnym zestawem informacji wejściowych jest pytanie w języku naturalnym oraz nazwy tabel i kolumn. Warte wzięcia pod uwagę są jednak także przynależności poszczególnych kolumn do tabel oraz relacje tworzone przez klucze podstawowe i obce. Jeżeli wykorzystywany został opisany wcześniej etap `schema linking`, to znalezione podczas niego relacje również powinny w tym momencie zostać zakodowane. Poniżej opisane zostały trzy popularniejsze drogi realizacji enkodowania, lecz jest ich znacznie więcej.

Najstarsza metoda enkodowania polega na zakodowaniu pytania naturalnego oraz nazw tabel i kolumn niezależnie od siebie, a następnie połączeniu tych informacji ze sobą na którymś etapie sieci neuronowej, która jest potem trenowana. Przykładami działających tak systemów jest `Seq2sql` (Zhong i in. 2017b) oraz `Sqlnet` (Xu i in. 2017). Wspomniane zakodowanie, czyli konwersja wartości tekstowych do reprezentujących je wektorów, nazywanych `embeddingami` (zanurzeniami), odbywa się w najprostszym przypadku poprzez wykorzystanie gotowych i publicznie dostępnych mapowań otrzymanymi algorytmami `Word2Vec` (Church 2017), czy też `GloVe` (Pennington i in. 2014).

Metodą pozwalającą osiągnąć lepsze wyniki od wcześniej wspomnianej jest serializacja informacji wejściowych do jednego długiego tekstu, który następnie jest enkodowany za pomocą pretrenowanego modelu językowego typu enkoder. Jest to zwykle model oparty na architekturze transformerów (Vaswani i in. 2017), gdzie `BERT` (Devlin i in. 2019) jest najczęstszym wyborem. Ich zaletą jest to, że poszczególne słowa enkodują w sposób kontekstowy, czyli na sposób kodowania każdego słowa mają wpływ wszystkie inne. Najbardziej problematyczne w tym podejściu jest to, że nie wszystkie informacje wejściowe da się łatwo zawrzeć w czystym tekście, na przykład relacje uzyskane w wyniku `schema linking`, czy klucze obce. W opisany sposób działa `IRNet` (Guo i in. 2019), czy analizowany w kolejnym rozdziale `BRIDGE` (Lin i in. 2020a).

Najbardziej elastyczna metoda, pozwalająca na wygodne zakodowanie wszystkich informacji wejściowych, opiera się na wykorzystaniu reprezentacji grafowej. Wówczas poszczególne tabele, kolumny oraz słowa z pytania interpretowane są jako węzły, a znane relacje pomiędzy nimi, w tym te wydobyte na etapie `schema linking`, tworzą krawędzie. Czynnikiem najbardziej hamującym powszechne wykorzystanie tej strategii jest trudność związana z przetwarzaniem grafów przez sieci neuronowe. Tą technikę enkodowania wykorzystuje testowany w kolejnej części model `RAT-SQL` (B. Wang i in. 2020b) oraz niedawno powstały `Graphix-T5` (J. Li i in. 2023).

5.4.3 Dekodowanie

Dekodowanie to finalny etap, którego celem jest zbudowanie gotowego zapytania SQL. Bazuje na uzyskanej w fazie enkodowania reprezentacji pośredniej i również jest realizowany z wykorzystaniem sieci neuronowych. Można wskazać trzy główne nurty, w które wpisują się tworzone rozwiązania i zostaną one pokrótce opisane.

Najbardziej ograniczoną strategią jest dekodowanie oparte na szkicach (ang. sketch-based). Zakłada ono przyjęcie pewnego szablonu zapytania SQL z lukami, które należy uzupełnić. W dużej mierze sprowadza to ten etap do zadania klasyfikacji, gdyż luki wymagają zwykle wyboru jednej z dostępnych tabel lub kolumn. Sprawdza się to jednak wyłącznie w uproszczonych scenariuszach, na przykład zakładających wykorzystanie tylko jednej tabeli, bez żadnych połączeń. Podejście to nie ma więc wielu praktycznych zastosowań i wydaje się być wykorzystywane głównie do poprawiania wyników osiąganych na prostych benchmarkach, jak zbiór `WikiSQL`. W ten sposób działające rozwiązania to `SqlNet` (Xu i in. 2017), `TypeSQL` (T. Yu, Z. Li i in. 2018), czy nowszy model zaproponowany przez Jianqiang Ma (Ma i in. 2020).

Bardziej popularną obecnie metodą dekodowania jest generowanie zapytania od zera, słowo po słowie. Dokonywane jest to poprzez uwzględnienie w konstruowanej sieci elementów rekurencyjnych, takich jak komórki `LSTM` (Y. Yu i in. 2019). Alternatywą jest douczenie jednego z pretrenowanych modeli językowych typu `enkoder-dekoder`. W rankingu zbioru `Spider` szczególnie popularny jest model `T5` (Raffel i in. 2020). Generowanie zapytań jako zwykłego tekstu ma ten problem, że mogą być one niepoprawne nawet pod względem strukturalnym, więc powstało wiele technik, by temu przeciwdziałać. Dekodowanie to wykorzystywane jest przez `Seq2sql` (Zhong i in. 2017b), `BRIDGE` (Lin i in. 2020a), czy `RESDSQL` (H. Li i in. 2023a).

Za jedne z najlepszych uważane są dekodery bazujące na gramatyce. Podobnie do poprzednich, generują zapytania od zera, element po elemencie, lecz z tą różnicą, że nie są to słowa. Na każdym kroku dekodowania produkują bowiem akcję, która służy do budowania drzewa AST reprezentującego zapytanie. Ma ono tą zaletę, że w każdym momencie jest poprawne, a po jego ukończeniu można przejść do zapytania w standardowej postaci. Dekodowanie oparte na gramatyce znalazło swoje miejsce w rozwiązaniach IRNet (Guo i in. 2019), czy też RAT-SQL (B. Wang i in. 2020b).

6 Eksperymenty

W niniejszej części przeprowadzone zostaną eksperymenty polegające na dostosowaniu, wytrenowaniu i przetestowaniu kilku modeli na stworzonych zbiorach. Do tego celu wybrane zostały rozwiązania RAT-SQL, BRIDE, RESDSQL oraz C3. Każde spośród nich zostanie przeanalizowane w osobnej części.

Modele RAT-SQL oraz BRIDGE to dość stare rozwiązania, które zostały wybrane ze względu na swoją popularność w literaturze oraz wykorzystanie odmiennych podejść. RESDSQL wyłoniono natomiast, gdyż jest to najwyżej znajdujący się w rankingu Spider model dedykowany, do którego dostępny jest kod źródłowy. C3 reprezentuje natomiast zupełnie inne podejście, gdyż bazuje na dużym modelu językowym od OpenAI. Jest to zatem ciekawe rozwiązanie, a koszty wynikające z wykorzystywanym przez nie modelem GPT-3.5-turbo są całkiem znośne.

W fazie eksperymentów trening modeli postanowiono wykonywać jedynie na zbiorze Spider. Cztery pozostałe będą wykorzystywane jedynie do przeprowadzania testów. Wynika to z chęci skrócenia czasu treningu oraz umożliwia dokonywanie uczciwych porównań z modelami trenowanymi przez innych badaczy, gdyż niemal zawsze jest to dokonywane właśnie na zbiorze Spider. Po fazie eksperymentów planuje się wybrać najbardziej obiecujące rozwiązanie i dokonać dłuższego treningu na wszystkich zbiorach.

Podczas treningu modeli uczenia maszynowego bardzo ważną rolę odgrywa posiadany sprzęt, a w przypadku sieci neuronowych najbardziej ograniczającym komponentem są karty graficzne. W czasie trwania pracy posiadano swobodny dostęp do dwóch jednostek. Pierwszą jest komputer stacjonarny wyposażony w kartę graficzną Nvidia RTX 2060 z 12 GB pamięci VRAM. Niestety posiadany procesor nie ma zintegrowanej karty graficznej, co powoduje, że wyświetlanie treści na monitorze zabiera prawie 2 GB i do wykorzystania dostępne jest jedynie 10 GB. Drugą jednostkę stanowi laptop Lenovo ThinkPad wyposażony w mobilną kartę Nvidia RTX 3080, posiadającą 16 GB pamięci, z czego całość jest dostępna do wykorzystania.

6.1 Model RAT-SQL

RAT-SQL to bardzo rozpoznawalny model, który był krzyżowany na przestrzeni czasu z różnymi innymi algorytmami i jego warianty można znaleźć w rankingu zbioru Spider na wielu pozycjach. Rozwiązanie to zostało zaproponowane w 2020 roku przez Bailin Wang oraz Richarda Shin (B. Wang i in. 2020b). Doczekało się dwóch kolejnych iteracji, określanych jako RAT-SQL v2

oraz RAT-SQL v3. W niniejszej pracy rozważana jest ta ostatnia, której kod dostępny jest na platformie GitHub w repozytorium Microsoftu (B. Wang i in. 2020a). Model ten produkuje zapytania bez wartości, co ogranicza jego praktyczne zastosowanie.

6.1.1 Działanie

Nazwa modelu RAT-SQL pochodzi od angielskiego wyrażenia *Relation Aware Transformer*, które doskonale opisuje to, co jest w tym rozwiązaniu najważniejsze. Wykorzystuje ono bowiem enkodowanie oparte na grafie, które zostało zrealizowane przy pomocy sieci typu transformer. Standardowo sieci takie potrafią analizować jedynie sekwencje, więc konieczne było zmodyfikowanie ich w taki sposób, aby działały również dla grafów. Dzięki dokonanym zmianom stały się one świadome relacji pomiędzy poszczególnymi elementami sekwencji.

Wszystkie sieci typu transformer posiadają element odpowiedzialny za uczenie się powiązań pomiędzy elementami wejściowymi w postaci mechanizmu uwagi (ang. attention). Autorzy RAT-SQL zbadali eksperymentalnie jego działanie dla problemu generowania zapytań SQL i zauważyli, że znajdywane połączenia są jednak dość słabe. Postanowili więc zmodyfikować mechanizm uwagi tak, aby poza znajdywaniem powiązań w standardowy, rozmyty sposób, można było jako dodatkowe wejście podawać powiązania z góry znane.

Relacje, które postanowiono jawnie zakodować poprzez zmodyfikowany mechanizm uwagi to przede wszystkim przynależność poszczególnych kolumn do tabel i powiązania tworzone przez klucze obce. Wykorzystano również etap schema linking w celu znalezienia połączeń pomiędzy fragmentami pytania i elementami bazy danych. Zrealizowano to poprzez wydobycie z pytania wszystkich n-gramów o długości od 1 do 5 i wyszukane ich dokładnych lub częściowych dopasowań wśród nazw tabel i kolumn. Połączenia typu value link znaleziono natomiast poprzez rozważenie każdego słowa z pytania osobno i wyszukiwanie go w każdej z dostępnych kolumn. Wszystkie te relacje tworzą skomplikowany graf, który RAT-SQL pozwala zakodować.

Przed przekazaniem wszystkich informacji wejściowych do transformera dokonującego enkodowania konieczna jest zamiana wartości tekstowych do postaci wektorowej. Uwaga ta tyczy się nazw tabel i kolumn oraz pytania. RAT-SQL pozwala dokonać takiej konwersji na dwa sposoby: z wykorzystaniem gotowych embeddingów nauczonych metodą GloVe lub pretrenowanym modelem BERT. Po enkodowaniu musi nastąpić dekodowanie, w którym wykorzystano rekurencyjne

komórki LSTM do generowania akcji, pozwalających zbudować drzewo AST. Gdy to nastąpi to drzewo zamieniane jest na tradycyjną postać zapytania SQL.

6.1.2 Modyfikacje dla języka polskiego

Przystosowanie RAT-SQL do języka polskiego okazało się dość wymagające ze względu na dużą bazę kodu. W pierwszej kolejności koniecznym było testowe uruchomienie tego rozwiązania, co już sprawiło problemy. W repozytorium autorów dostępny jest plik Dockerfile, teoretycznie umożliwiający proste wystartowanie środowiska, lecz budowa obrazu dockerowego okazała się zwracać błędy ze względu na wygaśnięte klucze GPG. Po naprawieniu tego problemu niezbędna była zmiana wersji kilku pakietów języka Python ze względu na niezgodności. W ramach tego zmieniono na bardziej aktualną bibliotekę pytorch, która jest wykorzystywana do tworzenia sieci neuronowych, gdyż starsza wydawała się nie wspierać architektury posiadanej karty graficznej.

Najważniejszą wśród dokonanych modyfikacji była zmiana mechanizmu tworzenia wektorowych reprezentacji tekstów na etapie enkodowania. W oryginalnej implementacji wykorzystany został w tym celu model BERT, a dokładnie wariant bert-large-uncased-whole-word-masking z platformy Hugging Face. Był on trenowany na tekstach angielskich, co go dla wersji polskiej dyskwalifikuje, a ponadto wprowadzana przez niego duża liczba parametrów sprawiała problemy z uruchomieniem na posiadanym sprzęcie. Postanowiono zastąpić go wielojęzycznym odpowiednikiem, trenowanym na 104 językach, oznaczonym na platformie HF identyfikatorem bert-base-multilingual-uncased. Posiada on trzykrotnie mniejszą liczbę parametrów, co umożliwia uruchomienie, lecz z pewnością wpływa na niższą skuteczność finalnego rozwiązania. Dostępne są również warianty trenowane tylko na języku polskim, lecz je wykluczono, ponieważ jak wcześniej zauważono, nazwy tabel i kolumn nawet w polskich bazach są najczęściej utrzymywane po angielsku – całkowicie polski wariant modelu BERT nie pokryłby tego przypadku.

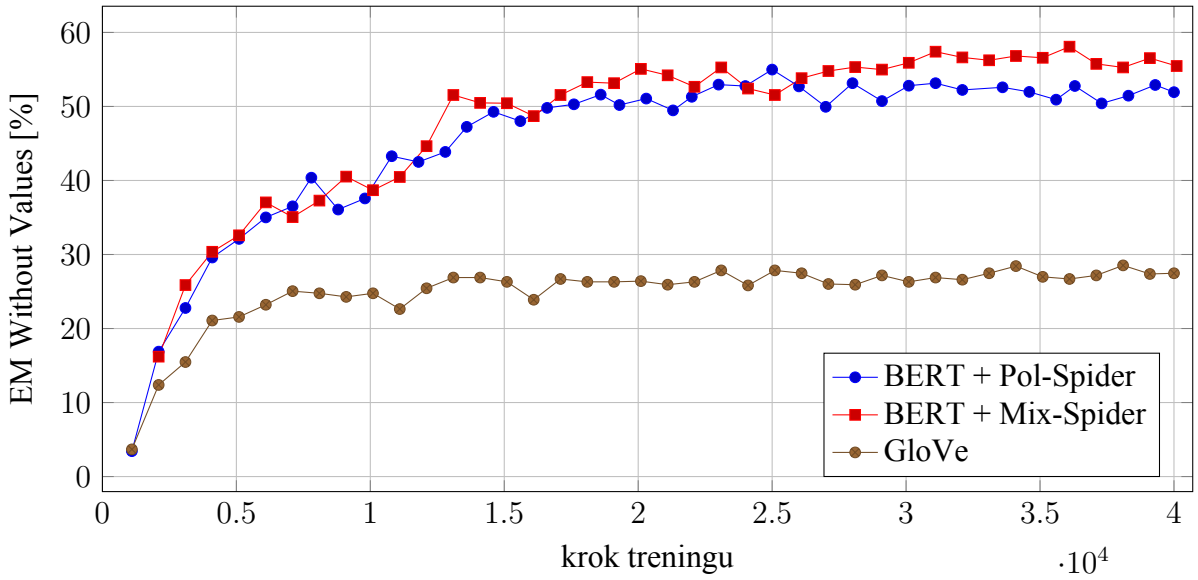
Jako alternatywną metodę tworzenia reprezentacji tekstów RAT-SQL wykorzystuje embeddingi nauczone metodą GloVe. Nie udało się niestety znaleźć dla nich odpowiednika wielojęzycznego, więc wykorzystano embeddingi całkowicie polskie, zamieszczone w repozytorium polish-nlp-resources (Dadas 2019). Dokładnie wybrano wariant 300d, w którym są one wektorami o długości trzystu elementów, ponieważ takiej długości były również oryginalnie zastosowanie embeddingi angielskie.

Wśród innych dokonanych zmian znalazła się modyfikacja listy słów nazywanych `stop words`, czyli powszechnie występujących, lecz nie niosących w sobie wiele informacji (i, w, z, na, do, się, o). Słowa takie są pomijane podczas poszukiwania częściowych dopasowań na etapie `schema linking` i należało podmienić je na polskie odpowiedniki. Ponadto w rozważanej bazie kodu kilka fragmentów wykorzystuje lematyzację, czyli zamianę słów na ich bazowe formy. W oryginalnej postaci zakłada ona pracę na tekstach angielskich, więc należało dostarczyć polską implementację, do czego wykorzystano bibliotekę `Stanza`. Ostatecznie etap `schema linking` bazował na znajdywaniu częściowych powtórzeń pomiędzy elementami bazy a fragmentami pytania. Jako że nazwy tabel i kolumn często nie zawierają polskich znaków, a pytania je posiadają, to słusznym działaniem wydało się zmodyfikowanie mechanizmu porównywania tak, aby ignorował różnicę w znakach specjalnych.

6.1.3 Eksperymenty

W ramach eksperymentów postanowiono wytrenować i przetestować model `RAT-SQL` w obu wariantach: z enkodowaniem tekstów za pomocą pretrenowanego modelu `BERT` oraz z enkodowaniem za pomocą embeddingów `GloVe`. W pierwszej konfiguracji model liczy 244 miliony parametrów, a w drugiej zaledwie 17 milionów, z czego wszystkie podlegają optymalizacji. Dla wariantu z `BERT` zdecydowano się wytrenować dwie wersje, gdzie jedna zostanie nauczona na polskim zbiorze `Pol-Spider`, a druga na zbiorze `Mix-Spider`, który zawiera dodatkowo oryginalny zbiór angielski. Niektóre artykuły podają bowiem, że modele trenowane na tego typu połączonych zbiorach tłumaczonych i angielskich osiągają lepsze wyniki niż trenowane jedynie na tłumaczeniach (Bakshandaeva i in. 2022; José i in. 2021), co postanowiono zweryfikować. Po wariancie z `GloVe` spodziewano się, że osiągnie słabsze wyniki od `BERT`, gdyż zgodnie z artykułem wprowadzającym `RAT-SQL` tak właśnie powinno się stać. Jak zostało wcześniej wspomniane, wykorzystane embeddingi `GloVe` trenowane były jedynie na tekstach polskich. Z tego powodu ten wariant postanowiono nauczyć dla bardziej ograniczonego scenariusza, w którym nazwy tabel i kolumn zawsze będą w języku polskim i trening przeprowadzono na zbiorze `Pol-Spider-PL`.

Autorzy `RAT-SQL` oryginalnie trenowali wariant `BERT` przez 80 000 kroków, co postanowiono skrócić o połowę. Pomimo tego równoległy trening na obu wspomnianych zbiorach, na dwóch posiadanych kartach graficznych, zajął prawie trzy doby. Wariant `GloVe` trenowany był natomiast oryginalnie przez 40 000 kroków, co postanowiono pozostawić bez zmian i w tym przypadku zajęło to niecałe dwie doby.



Rys. 6.1: Wyniki modeli na zbiorze testowym w trakcie trwania treningu. *Warianty `BERT` testowano na zbiorze `Pol-Spider`, natomiast `GloVe` na zbiorze `Pol-Spider-PL`.*

Podczas treningu wagi modeli były regularnie zapisywane na dysku w odstępie 1000 kroków. Po zakończeniu każdy taki zestaw wag został przetestowany z wykorzystaniem metryki `EM Without Values`, co zajęło sumarycznie kilkanaście kolejnych godzin obliczeń. Warianty z `BERT` testowane były na zbiorze `Pol-Spider`, natomiast wariant z `GloVe` na zbiorze `Pol-Spider-PL`. Wykres przedstawiający zmianę skuteczności modeli wraz z kolejnymi krokami treningu przedstawiono na rysunku 6.1. Ostatecznie z każdego z trzech zestawów wag wybrano do dalszych testów ten, który podczas przedstawionej ewaluacji otrzymał najlepsze rezultaty.

6.1.4 Wyniki

Trzy wytrenowane modele przetestowano na różnych konfiguracjach zbioru `Spider`. Wyniki tej analizy przedstawiono w tabeli 6.1. Można z niej odczytać, że w każdym przypadku najlepiej nauczył się wariant `BERT` trenowany na zbiorze `Mix-Spider`, nieco niższą skuteczność osiągnął `BERT` uczony na zbiorze `Pol-Spider`, natomiast model wykorzystujący embeddingi `GloVe` znacznie od tych dwóch odstaję. Z wykorzystaniem najlepszego modelu przeprowadzono dalsze testy, w których dokonano jego ewaluacji na wszystkich zbiorach pokrewnych, z podziałem na poziomy trudności zapytań. Wyniki tych testów przedstawiono w tabeli 6.2.

Wariant	zbiór treningowy	Zbiór testowy			
		Pol-Spider	Pol-Spider-PL	Pol-Spider-EN	En-Spider
BERT	Pol-Spider	53,1	56,6	49,7	42,8
BERT	Mix-Spider	58,1	61,0	55,1	68,8
GloVe	Pol-Spider-PL	19,3	27,5	11,2	2,8

Tabela 6.1: Wyniki modeli otrzymane metodą RAT-SQL na wariantach zbioru Spider. *Wartości w komórkach tabeli to wyniki metryki EM Without Values wyrażone procentowo.*

Zbiór	Easy	Medium	Hard	Extra Hard	Razem
Pol-Spider	73,8	58,7	50,9	40,4	58,1
Pol-Spidersyn	58,5	45,1	41,2	26,9	44,5
Pol-Spiderdk	60,9	36,4	31,1	17,6	37,0
Pol-Sparc	62,1	42,2	10,0	25,0	53,6
Pol-Cosql	59,6	47.5	22,1	29,4	48,7

Tabela 6.2: Wyniki najlepszego modelu RAT-SQL na zbiorach pokrewnych. *Najlepszym modelem jest wariant BERT uczony na zbiorze dwujęzycznym. Wartości w komórkach tabeli to wyniki metryki EM Without Values wyrażone procentowo.*

6.1.5 Analiza

Istotną obserwacją jest to, że model trenowany na połączeniu zbioru polskiego i angielskiego nauczył się lepiej przewidywać zapytania SQL od tego uczonego wyłącznie na zbiorze polskim. Jest to zgodne z eksperymentami przeprowadzonymi i opisanymi również przez innych badaczy, które podsumowano w tabeli 6.3. Można z niej odczytać rezultaty, które udało się osiągnąć twórcom dwóch innych tłumaczeń zbioru Spider, którzy także eksperymentowali z metodą RAT-SQL i nauką na różnych wariantach swoich zbiorów. W przypadku zbioru portugalskiego zysk otrzymany na poszerzonym zbiorze jest dość niewielki, natomiast dla zbioru rosyjskiego wynosi aż 6 punktów procentowych. Korzyść dla tłumaczenia polskiego jest dość duża i bliższa temu ostatniemu.

Tłumaczenie zbioru Spider	Zbiór treningowy	
	Tłumaczenie	Tłumaczenie + angielski
Rosyjskie (PAUQ)	51,0	57,0
Portugalskie	58,8	59,5
Polskie (Pol-Spider)	53,1	58,1

Tabela 6.3: Wyniki osiągane przez RAT-SQL dla różnych zbiorów treningowych. *Dane dotyczą modelu w wariancie BERT. Wartości w komórkach tabeli to wyniki metryki EM Without Values wyrażone procentowo.*

Wyniki osiągnięte przez model w wariancie z GloVe okazały się być znacznie niższe w porównaniu z wariantami BERT. Było to spodziewane, biorąc pod uwagę choćby liczbę parametrów, jednak wydaje się, że wyniki metryk mogłyby być nieco wyższe. Wytrenowany przez twórców RAT-SQL model odstaję od wariantu BERT o około 7 punktów procentowych, a w naszym przypadku różnica wynosi ponad 20 punktów. Powodem tego może być przypuszczalnie gorsza jakość embeddingów GloVe dla języka polskiego niż angielskiego, lecz dalszych eksperymentów zaniechano. Model w tym wariancie mógłby stanowić nieco słabszą alternatywę, którą można szybko wytrenować i uruchamiać na mniej wydajnych urządzeniach. Aktualna dokładność wydaję się jednak za niska, aby to znalazło praktyczne zastosowanie.

Dane liczbowe w tabeli 6.2 pokazują, że wraz ze zwiększaniem się poziomu trudności zapytań maleje skuteczność modelu, co jest spodziewanym zachowaniem. Niewielkie rozbieżności od tej reguły można zaobserwować jedynie dla zbiorów Pol-Cosql oraz Pol-Sparc. Przyczyną tego jest zapewne niewielka liczba znajdujących się w nich trudnych zapytań, przez co kilka pozornie wymagających mogło znacząco zachwiać wynikami.

Efekty uzyskane na zbiorze Pol-Spidersyn okazały się wyraźnie słabsze w porównaniu do tych na Pol-Spider, bo różnica wynosi aż 13,57 punktów procentowych. Potwierdza to obserwację przedstawioną w artykule wprowadzającym Spider-Syn. Mówi ona, że modele mają duże problemy z odpowiadaniem na pytania, w których użytkownik nie znając dokładnie schematu bazy danych, posługuje się synonimami.

Niskie wyniki, chociaż również spodziewane, osiągnął model na zbiorze Spider-DK. Zawiera on bowiem pytania wymagające znajomości wiedzy domenowej, z którą uważa się, że modele sobie często nie radzą. Spadek, który nastąpił w tym przypadku względem zbioru Pol-Spider, wynosi 21,07 punktów procentowych. W celu dokonania głębszej analizy możliwe jest obliczenie dokładności modelu w obrębie każdego typu wiedzy domenowej z osobna, czego jednak postanowiono nie robić.

6.2 Model BRIDGE

BRIDGE to rozwiązanie, które zostało opublikowane w roku 2020 przez Xi Victorię Lin, Richarda Sochera oraz Caiming Xionga (Lin i in. 2020a). Powstało więc w podobnym czasie jak RAT-SQL, ale w odróżnieniu od niego generuje kompletne zapytania SQL z wartościami, co jest istotną zaletą. Jego kod źródłowy jest umieszczony na platformie GitHub (Lin i in. 2020b).

6.2.1 Działanie

Działanie BRIDGE znacząco odbiega od RAT-SQL. W tym przypadku wykorzystano enkodowanie oparte na serializacji informacji wejściowych do długiego tekstu i przekazywaniu go do pretrenowanego modelu BERT. Dekodowanie opiera się natomiast na generowaniu słów jedno po drugim, zamiast generowania akcji tworzących drzewo AST. Na obu etapach zastosowano ciekawe techniki, które dodatkowo poprawiają skuteczność.

Enkodowanie opiera się na skonstruowaniu sekwencji tekstowej zawierającej wszystkie nazwy tabel, gdzie po każdej nazwie tabeli znajduje się lista zawartych w niej kolumn. Na początku tej sekwencji doklejane jest dodatkowo rozpatrywane pytanie. Wcześniej wykonywany jest etap schema linking w celu znalezienia połączeń typu value link i odnalezione dopasowania w zawartości bazy danych są wstawiane po nazwie odpowiedniej kolumny. W celu zachowania znaczenia poszczególnych części stworzonej sekwencji wykorzystywane są specjalne tokeny [T], [C] oraz [V], które są wstawiane odpowiednio przed każdą nazwą tabeli, kolumny i wartością. Stworzony tekst jest następnie przetwarzany przez model językowy BERT i dodane po nim dwie lekkie warstwy LSTM. W ten sposób uzyskiwane są wykorzystywane przez dekoder wektorowe reprezentacje wszystkich tabel, kolumn i pytania. Reprezentacje kolumn są jednak dodatkowo wzbogacane za pomocą trenowanych równolegle wektorów reprezentujących metainformacje, takie jak bycie kluczem podstawowym, bycie kluczem obcym, czy posiadanie konkretnego typu danych. Łącznie podstawowych reprezentacji z tymi metainformacjami dokonuje się prostą warstwą liniową.

Wykorzystany dekoder generuje wyjściowe zapytanie słowo po słowie. Nie jest to jednak typowy dekoder, jak te wykorzystywane w modelach językowych, ponieważ na każdym kroku dekodowania, poza wyprodukowaniem jednego tokena ze słownika, może dokonać także kopiowania go z pytania lub spośród nazw tabel i kolumn. Aby umożliwić takie zachowanie standardowy dekoder, bazujący na LSTM, został połączony z siecią typu Pointer Network (Vinyals i in. 2015), która potrafi wskazywać konkretne pozycje w sekwencjach wejściowych. Poza tym dekoder został nauczony w taki sposób, aby produkować zapytania w kolejności wykonywania, czyli takiej, w której wykonałby je silnik bazodanowy. Powoduje to, że nazwy tabel generowane są przed nazwami kolumn i dzięki temu podczas generowania nazwy kolumny można ograniczyć się jedynie do kolumn dostępnych we wcześniej wymienionych tabelach.

6.2.2 Modyfikacje dla języka polskiego

Przystosowanie BRIDGE do języka polskiego okazało się znacznie prostsze niż to było w przypadku RAT-SQL. Tutaj również pracę rozpoczęto od uruchomienia oryginalnego rozwiązania, więc napisano plik Dockerfile, który tworzy obraz dockerowy zawierający kompletne środowisko. Jedynym problemem okazał się brak wśród zależności projektu biblioteki numpy, która była wykorzystywana. Naprawiono to poprzez dodanie odpowiedniej linii kodu.

Jedyną ważną modyfikacją dla języka polskiego okazała się zmiana wykorzystywanego modelu BERT z wariantu bert-large-uncased na bert-base-multilingual-uncased, oba dostępne są na platformie HF. Jest to działanie analogiczne do zmiany zaaplikowanej w RAT-SQL. Uzasadnienie również jest podobne: zmieniono model na wielojęzyczny, by poradził sobie z językiem polskim oraz na mniejszy, by umożliwić naukę na posiadanym sprzęcie. Tak zmodyfikowany model BRIDGE liczy sobie 174 miliony parametrów i wszystkie podlegają treningowi.

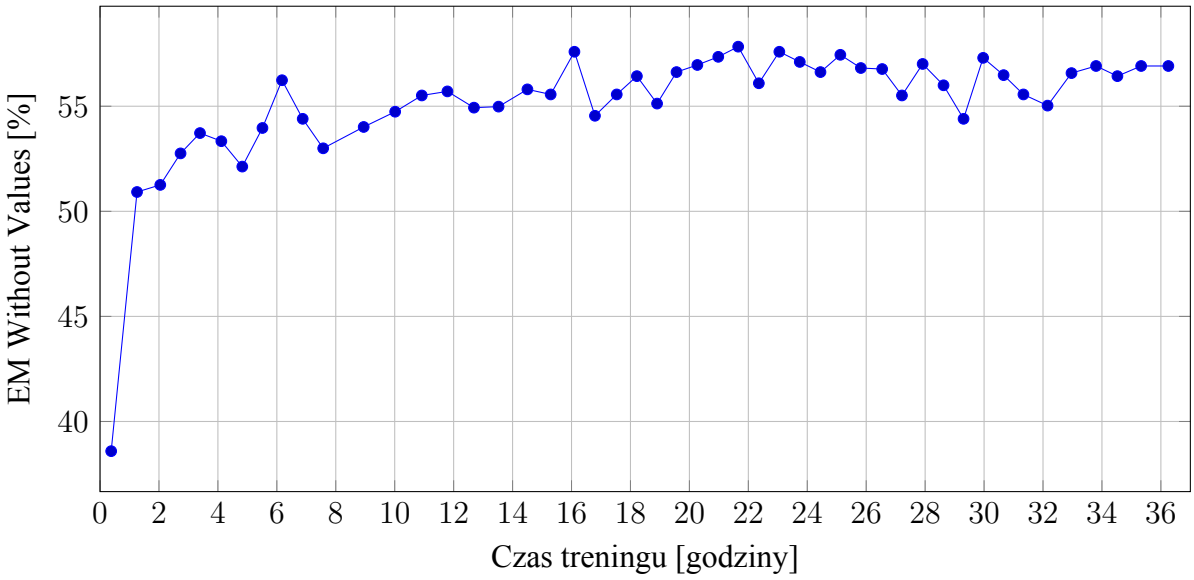
6.2.3 Eksperymenty

W ramach eksperymentu chciano dokonać treningu modelu BRIDGE i go przetestować. Naukę przeprowadzono na zbiorze Mix-Spider, ponieważ wcześniej przeprowadzone eksperymenty z RAT-SQL pokazały, że model trenowany na połączonym zbiorze polskim i angielskim nauczył się lepiej. Oczywiście nie można założyć na tej podstawie, że każdy model uczony na połączonym zbiorze będzie skuteczniejszy. Można jednak przypuszczać, że zwykle tak jest.

Model uczono przez 36 godzin na jednostce wyposażonej w kartę Nvidia RTX 3080, a zaimplementowany wewnątrz procedury treningu kod dokonywał w tym czasie regularnego obliczania metryki EM Without Values na części testowej zbioru Pol-Spider. Wykres przedstawiający zmianę wartości tej metryki na przestrzeni czasu przedstawiono na rysunku 6.2. Na wykresie widać, że skuteczność modelu rosła wyraźnie przez połowę czasu treningu, a w drugiej połowie już nie widać tendencji wzrostowej – dlatego naukę postanowiono przerwać. Jako finalny został wybrany model z punktu, w którym wartość wspomnianej metryki była najwyższa.

6.2.4 Wyniki

Wyniki przeprowadzonego eksperymentu przedstawiono w tabeli 6.4. Jest ona znacznie bardziej rozbudowana niż to było w przypadku rozwiązania RAT-SQL, ponieważ ten model zwraca zapytania z wartościami. Możliwe więc stało się obliczenie metryk EM oraz EX.



Rys. 6.2: Wyniki modelu BRIDGE w czasie trwania treningu. *Model sprawdzano na części testowej zbioru Pol-Spider.*

Zbiór	Easy	Medium	Hard	Extra Hard	Razem
Pol-Spider	79,4/71,2/82,9	59,8/51,9/68,9	43,1/40,2/63,8	34,6/31,9/54,5	57,6/51,4/69,1
Pol-Spider-PL	80,6/72,6/84,3	60,3/52,5/70,0	44,8/40,2/64,9	34,9/31,3/53,6	58,5/51,8/69,9
Pol-Spider-EN	78,2/69,8/81,5	59,2/51,3/67,9	41,4/40,2/62,6	34,3/32,5/55,4	56,8/50,9/68,3
En-Spider	86,7/82,3/85,1	67,9/62,8/71,7	51,7/48,3/58,0	40,4/38,0/43,4	65,3/61,0/68,1
Pol-Spidersyn	63,2/56,1/71,6	48,3/41,7/59,3	37,1/36,4/52,6	23,1/21,5/43,8	45,7/40,8/58,4
Pol-Spiderdk	58,6/51,4/63,6	39,4/32,9/49,4	22,3/22,3/44,6	15,7/13,8/31,9	36,4/31,5/48,2
Pol-Sparc	61,7/60,0/68,1	38,8/44,1/60,2	3,3/25,0/43,3	25,0/17,6/43,8	52,0/47,6/64,3
Pol-Cosql	60,0/52,5/69,2	44,1/34,7/63,6	25,0/19,1/54,4	17,6/14,7/47,1	47,6/40,2/63,9

Tabela 6.4: Wyniki modelu BRIDGE na poszczególnych zbiorach. *Wartości w każdej komórce posiadają format EM Without Values / EM / EX i zostały wyrażone w procentach.*

6.2.5 Analiza

Zestawiając ze sobą wyniki osiągnięte metodą BRIDGE oraz RAT-SQL trudno jest wskazać pod względem metryki EM Without Values, która jest jedyną wspólną, model lepszy. Na częściach testowych wszystkich wariantów zbioru Spider analizowany teraz model BRIDGE okazuje się działać lepiej. Na zbiorach pokrewnych Pol-Spiderdk, Pol-Sparc oraz Pol-Cosql jednak to RAT-SQL jest skuteczniejszy. Można z tego wyciągnąć wniosek, że BRIDGE sprawdza się lepiej,

jeżeli trenowany i testowany jest na podobnych danych. RAT-SQL wydaje się natomiast posiadać bardziej rozwiniętą umiejętność generalizacji i wykorzystania wiedzy domenowej, przez co osiąga lepsze wyniki na zbiorach wyraźnie odbiegających od treningowego.

Metoda BRIDGE nie jest już tak popularna w literaturze jak RAT-SQL i eksperymenty z jej wykorzystaniem przeprowadzili jedynie autorzy rosyjskiego tłumaczenia zbioru Spider, czyli autorzy PAUQ. Zestawienie osiągniętych przez nich wyników z niniejszymi przedstawiono w tabeli 6.5. Widać, że zgodnie z metryką EM Without Values na zbiorze polskim udało się osiągnąć rezultaty nieco lepsze. Możliwą przyczyną jest przypuszczalnie większa liczba danych polskich, na których wielojęzyczny model BERT mógł być trenowany, albo też wyższy stopień skomplikowania języka rosyjskiego. Różnica w wynikach metryki EX jest natomiast bardzo duża, na korzyść zbioru polskiego. Przyczyną tego jest pewne zakłamanie metryki EX dla języka polskiego, o czym wcześniej wspomniano. W pełni uzasadnione jej zaaplikowanie wymagałoby bowiem przetłumaczenia zawartości wszystkich baz danych, czego dla polskiego zbioru nie zrobiono, a w przypadku PAUQ zostało dokonane częściowo. Szczególnie więc w tym przypadku, ze względu tłumaczenia zawartości baz w tylko jednym zbiorze, należy uważać z porównywaniem tych metryk ze sobą. Mimo wszystko przedstawione zestawienie stanowi potwierdzenie tego, że modyfikacji metody BRIDGE dla języka polskiego dokonano poprawnie.

Tłumaczenie zbioru Spider	**Zbiór treningowy**	
	Tłumaczenie	**Tłumaczenie + angielski**
Rosyjskie (PAUQ)	52,0 / 48,0	55,0 / 50,0
Polskie (Pol-Spider)	—— / ——	57,6 / 69,1

Tabela 6.5: Zestawienie wyników modelu BRIDGE dla polskiego i rosyjskiego tłumaczenia. *Komórki tabeli zawierają wyniki metryk w formacie EM Without Values / EX, są wyrażone w procentach.*

Analizując dalej tabelę 6.4, można zauważyć, że wyniki metryki EM Without Values są zawsze większe od EM, co z jednej strony jest oczywiste, ale warto to podkreślić. Różnica pomiędzy nimi wynosi średnio 5,6 punktów procentowych, co oznacza, że ponad pięć procent generowanych zapytań jest poprawnych pod względem struktury, lecz posiada błąd w przewidzianych wartościach. Widać więc, że generowanie wartości stanowi istotne wyzwanie. Dla RESDSQL aktualne są spostrzeżenia poczynione dla poprzedniego modelu, takie jak spadek skuteczności wraz ze wzrostem poziomu trudności, czy niższa dokładność w przypadku zbiorów pokrewnych.

6.3 Model RESDSQL

RESDSQL to dość nowe rozwiązanie, gdyż powstało na początku 2023 roku. Zostało opublikowane w artykule *RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL* (H. Li i in. 2023a). Na chwilę pisania niniejszej pracy jest to najwyżej znajdujące się w rankingu Spider rozwiązanie, które nie opiera się na wykorzystaniu dużych pretrenowanych modeli językowych od OpenAI i do którego dostępny jest kod źródłowy (H. Li i in. 2023b).

6.3.1 Działanie

RESDSQL bazuje w dużej mierzę na pretrenownym modelu językowym T5 (Raffel i in. 2020). Jest on kompletnym modelem typu transformer, który służy do generowania tekstowych odpowiedzi na podstawie tekstowych informacji wejściowych. Został już wcześniej wytrenowany na obszernym zbiorze, więc zawiera dużą ilość wiedzy. RESDSQL dokonuje dotrenowania tego modelu tak, aby zwracał zapytania SQL, będące odpowiedzią na podawane pytania. Podczas konstruowania sekwencji wejściowej oraz dekodowania sekwencji wyjściowej wprowadza jednak ciekawe techniki, które poprawiają jego skuteczność dla rozważanego problemu Text-to-SQL. Ich głównym celem jest rozdzielenie od siebie generowania szkieletu zapytań i uzupełniania go konkretnymi nazwami tabel i kolumn, co z resztą zostało podkreślone w tytule artykułu.

Ważnym elementem, odróżniającym RESDSQL od wcześniej opisanych rozwiązań jest to, że do enkodera zawartego wewnątrz T5 nie są przekazywane wszystkie nazwy tabel i kolumn, lecz tylko te, które zostały uznane w kontekście danego pytania za najbardziej istotne. Celem tego działania jest odciążenie enkodera z wykonywania skomplikowanego procesu schema linking. Aby dostać informację o istotności poszczególnych tabel i kolumn, dla podanego pytania, trenowana jest wcześniej całkowicie niezależna sieć, która przyjmuje pytanie oraz wszystkie nazwy tabel i kolumn i dla każdej z nich zwraca prawdopodobieństwo, które może być interpretowane jako stopień istotności. Sieć ta określana jest przez swoich autorów mianem cross-encoder. Ostatecznie do modelu T5 przekazywane są 4 najistotniejsze tabele, a dla każdej z nich 5 najważniejszych kolumn.

Z punktu widzenia dekodowania największą nowością jest to, że model T5 nie jest dotrenowywany tak, aby od razu produkować kompletne zapytania, lecz na początku zwrócić szkielet, a dopiero za nim wykonywalne zapytanie. Uzasadnieniem opisanego zachowania jest to, że początkowe przewidzenie szkieletu jest w miarę prostym problemem, więc powinno mieć dużą dokładność.

Dekodery w modelach językowych posiadają własność nazywaną autoregresyjnością, która polega na tym, że podczas generowania kolejnych fragmentów odpowiedzi brany jest pod uwagę tekst wygenerowany do tej pory. Dzięki temu model T5 podczas produkowania drugiej części odpowiedzi, która zawiera kompletne zapytanie SQL, może odwoływać się do wcześniejszej części z szablonem i traktować ją jako pewnego rodzaju notatnik.

Ciekawą techniką, z którą RESDSQL można łączyć, jest NatSQL (Gan, Chen, Xie i in. 2021). Wprowadza ona alternatywną reprezentację dla zapytań SQL, która bardziej przypomina język naturalny i w związku z tym jest łatwiejsza do nauczenia i generowania przez większość modeli. Jednocześnie jest ona na tyle jednoznaczna, że można przekonwertować ją na tradycyjne zapytania SQL. Zgodnie z eksperymentami autorów RESDSQL zastąpienie tradycyjnych zapytań za pomocą NatSQL pozwoliło na zbiorze Spider zwiększyć skuteczności mierzoną metryką EM o około 2 punkty procentowe.

6.3.2 Modyfikacje dla języka polskiego

Rozwiązanie RESDSQL okazało się wyjątkowo proste do przystosowania dla języka polskiego, bo nie trzeba było wykonywać żadnych kreatywnych modyfikacji. Metoda ta została bowiem już wcześniej wykorzystana do pracy na rosyjskim zbiorze PAUQ i stosowny kod znajdywał się w repozytorium. Wystarczyło dodać kilka nowych skryptów, które nieznacznie różnią się od istniejących.

Najbardziej istotną modyfikacją, niezbędną do nauki na polskim, czy też rosyjskim tłumaczeniu, jest zmiana wykorzystywanego modelu T5 na mT5 (Xue i in. 2021). Różnica pomiędzy nimi jest jedynie taka, że pierwszy został nauczony na tekstach angielskich, natomiast drugi na zbiorze zawierającym 101 różnych języków, w tym wymienione dwa.

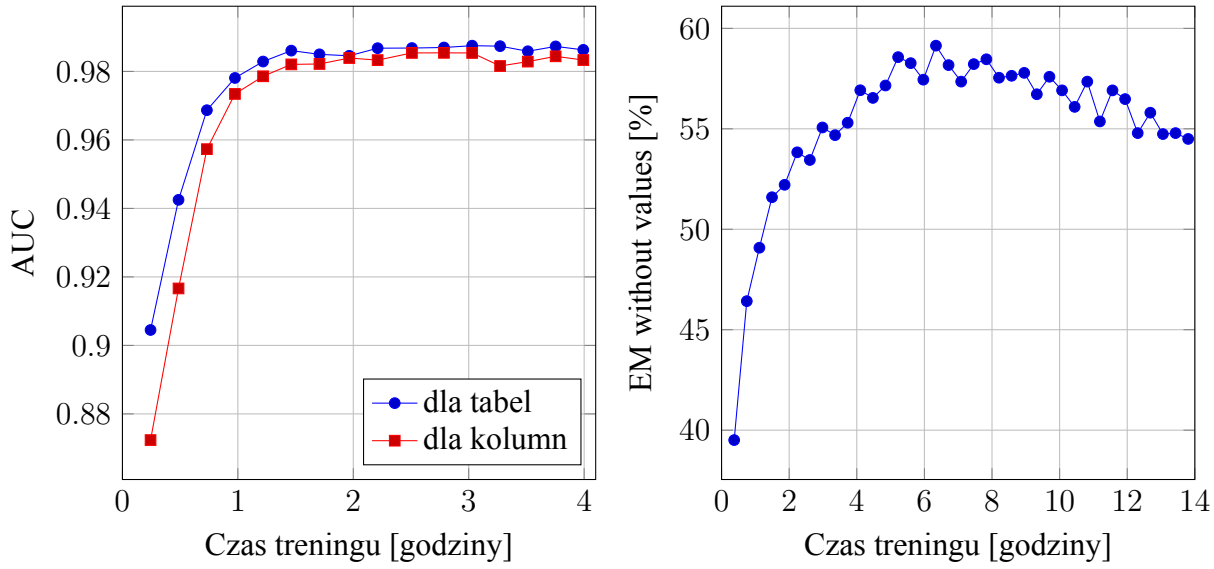
Okazuje się, że niestety dla języka polskiego nie można łatwo wykorzystać obiecującego połączenia niniejszej metody z NatSQL. Przyczyną jest brak udostępnienia przez autorów NatSQL skryptu przekształcającego tradycyjne zapytania SQL do zaprojektowanej przez nich postaci. Opublikowane są jedynie zapytania ze zbioru Spider po dokonaniu takiej konwersji. W przypadku rosyjskiego tłumaczenia udało się zastosować NatSQL, lecz wymagało to przekształcenia każdego zapytania w sposób ręczny. Dla zbioru polskiego należałoby postąpić analogicznie, co wymagałoby wiele żmudnej pracy, której postanowiono uniknąć.

6.3.3 Eksperymenty

W ramach eksperymentu postanowiono dokonać nauki RESDSQL na zbiorze Mix-Spider, ponieważ, jak wcześniej zauważono, nauka na dwujęzycznych zbiorach wydaje się mieć lepsze efekty niż na jednojęzycznych. Konieczny był wybór konkretnego wariantu dotrenowywanego modelu mT5, ponieważ występuje on w kilku rozmiarach. Na mocniejszej z dwóch posiadanych kart graficznych, czyli RTX 3080, udało się uruchomić jedynie wariant najmniejszy – mt5-base. Sprawdzając poszczególne rozmiary, stosowano minimalną wielkość wsadu (ang. batch size), czyli ilość danych jednocześnie przekazywanych do modelu, aby zmniejszyć zapotrzebowanie na pamięć VRAM. Po dokonaniu głębszej ingerencji w procedurę treningu udało się ostatecznie uruchomić ją także dla większego modelu. Wymagało to jednak załadowania wag i treningu z wykorzystaniem arytmetyki 16-bitowej zamiast powszechnie stosowanej 32-bitowej. Może się to wiązać z pojawieniem podczas nauki różnych niestabilności, więc z powodu braku doświadczenia z tego typu treningiem postanowiono pozostać przy oryginalnym.

Zgodnie z wcześniejszym opisem wykorzystanie RESDSQL wymaga w pierwszej kolejności wytrenowania sieci cross-encoder, która dokonuje oceny istotności tabel i kolumn. Trening ten wykonywano przez 4 godziny, a wykres przedstawiający zmianę skuteczności modelu w jego trakcie umieszczono na rysunku 6.3a. Wspomniana skuteczność jest tutaj wyrażana za pomocą metryki AUC (Ling i in. 2003), spotykanej w zadaniach klasyfikacji. Po 4 godzinach trening został automatycznie przerwany, ponieważ od dłuższego czasu nie nastąpiło zwiększenie skuteczności rozumianej jako sumy AUC dla tabel i kolumn. Jako produkt tego etapu został wybrany model z zestawem wag, dla którego metryka ta była najwyższa.

Drugim etapem nauki było dotrenowywanie modelu T5 dla zadania generowania zapytań SQL. Tę część treningu zakończono po 14 godzinach. Tak jak jest to ukazane na rysunku 6.3b, przez pierwsze 7 godzin dokładność generowanych zapytań rosła, lecz potem zaczęła regularnie spadać i w godzinie 14 osiągnęła na tyle niską wartość, że dalszego treningu zaprzestano. Przyczyną tego mogło być przetrenowanie modelu (Hawkins 2004), czyli zbytnie przystosowanie do danych treningowych. Jako finalny został wybrany model z punktu, w którym wartość rozważanej metryki była najwyższa.



(a) model oceniający istotność tabel i kolumn

(b) model przewidujący zapytania

Rys. 6.3: Wyniki modeli składowych RESDSQL w czasie trwania treningu. *Modele sprawdzano na części testowej zbioru Pol-Spider.*

6.3.4 Wyniki

W tabeli 6.6 przedstawione zostały wyniki przeprowadzonego eksperymentu. Rozważany model, czyli RESDSQL, podobnie jak poprzedni, generuje kompletne zapytania SQL z wartościami. Możliwe więc było obliczenie metryk EM oraz EX.

Zbiór	Easy	Medium	Hard	Extra Hard	Razem
Pol-Spider	76,2/70,6/83,5	61,9/56,3/73,1	50,0/46,0/62,9	35,8/30,4/53,3	59,1/53,8/70,7
Pol-Spider-PL	78,6/72,6/85,1	64,1/58,5/74,4	50,0/44,8/62,1	32,5/27,1/50,0	60,2/54,5/71,0
Pol-Spider-EN	73,8/68,5/81,9	59,6/54,0/71,7	50,0/47,1/63,8	39,2/33,7/56,6	58,1/53,1/70,4
En-Spider	81,5/79,4/86,3	69,3/66,4/75,8	51,7/50,6/65,5	47,0/45,8/50,0	65,7/63,5/72,4
Pol-Spidersyn	61,7/57,6/72,5	52,2/48,7/65,8	42,6/41,9/57,0	26,4/21,9/44,6	48,6/45,3/62,4
Pol-Spiderdk	52,7/48,2/62,3	34,8/31,5/50,2	20,9/20,9/37,8	17,6/13,3/32,4	33,2/29,9/47,5
Pol-Sparc	61,3/59,0/72,3	38,3/33,0/60,2	13,3/13,3/43,3	43,8/43,8/50,0	52,5/49,5/67,2
Pol-Cosql	61,7/56,7/71,3	54,2/48,3/63,6	26,5/25,0/51,5	20,6/20,6/55,9	51,5/47,2/65,2

Tabela 6.6: Wyniki modelu RESDSQL na poszczególnych zbiorach. *Wartości w każdej komórce posiadają format EM Without Values / EM / EX i zostały wyrażone w procentach.*

6.3.5 Analiza

W porównaniu z dwoma wcześniej analizowanymi rozwiązaniami RESDSQL wydaje się wypadać najlepiej. Inaczej mówią jedynie wyniki dwóch testów. W przypadku zbiorów Pol-Sparc oraz Pol-Spiderdk, patrząc na metrykę EM, najlepiej wypadł bowiem model pierwszy, czyli RAT-SQL. W pozostałych przypadkach to jednak RESDSQL dominuje.

Warto zwrócić uwagę na fakt, że tak dobre wyniki udało się osiągnąć w stosunkowo niedługim czasie. Pomimo tego, że nauka odbywała się w sposób dwuetapowy, to sumaryczny czas, po którym modele osiągnęły maksymalną skuteczność, okazał się dla RESDSQL najkrótszy spośród rozwiązań do tej pory rozważanych.

Tak jak zostało wcześniej zauważone, RESDSQL jest dość świeżym rozwiązaniem, które zostało opublikowane już po stworzeniu wszystkich istniejących tłumaczeń zbioru Spider, więc ich twórcy nie mieli nawet możliwości wykonać na RESDSQL eksperymentów. Autorzy tej metody jakiś czas od wydania swojego artykułu wyszli jednak z inicjatywą i postanowili dokonać treningu i testów swojego modelu na chińskim tłumaczeniu, czyli zbiorze CSpider. Wyniki zamieścili w repozytorium na platformie GitHub. W tabeli 6.7 zestawiono wyniki osiągnięte z wykorzystaniem modelu RESDSQL na zbiorze polskim i chińskim. Na tym ostatnim wytrenowanych zostało kilka modeli, więc wybrano ten, który odpowiada polskiemu pod względem konfiguracji.

Tłumaczenie zbioru Spider	**Zbiór treningowy**	
	Tłumaczenie	**Tłumaczenie + angielski**
Chińskie (CSpider)	71,7 / 77,9	— / —
Polskie (Pol-Spider)	— / —	59,1 / 70,7

Tabela 6.7: Zestawienie wyników modelu RESDSQL dla polskiego i chińskiego tłumaczenia. *Komórki tabeli zawierają wyniki metryk w formacie EM Without Values / EX, podane procentowo.*

Zgodnie z zawartymi w tabeli 6.7 danymi liczbowymi na chińskim zbiorze udało się osiągnąć wartość metryki EM Without Values większą o ponad 10 punktów procentowych niż dla zbioru polskiego. Do tego treningu dokonano wyłącznie na tekstach chińskich, bez dołączania zbioru angielskiego, co przypuszczalnie powinno dać jeszcze lepsze rezultaty. Duża różnica pomiędzy wynikami otrzymanymi na polskim i rosyjskim tłumaczeniu jest dość zastanawiająca i pierwszym przypuszczeniem było to, że wykorzystywany w obu przypadkach wielojęzyczny model mT5 był trenowany na zbiorze zawierającym większą ilość tekstów w języku chińskim niż polskim. Załącznik do artykułu wprowadzającego mT5 (Xue i in. 2021) ujawnia jednak, że było

wręcz odwrotnie – w zbiorze treningowym znalazło się więcej danych polskich niż chińskich. Przypuszczenie to zostało więc odrzucone. Innym powodem mogła być lepsza jakoś zbioru chińskiego, który został przetłumaczony manualnie, co ułatwiło modelowi naukę.

6.4 Model C3

C3 jest dość nowym rozwiązaniem, które zostało opublikowane w lipcu 2023 roku w artykule pod tytułem *C3: Zero-shot Text-to-SQL with ChatGPT* (Dong i in. 2023b). Należy ono do niedawno powstałego nurtu zaprzęgającego duże modele językowe do generowania zapytań SQL. Zapytania są kompletne, gdyż model ten przewiduje również odpowiednie wartości. Cały kod źródłowy C3 został udostępniony przez autorów za pośrednictwem platformy GitHub (Dong i in. 2023a).

6.4.1 Działanie

Sposób działania C3 istotnie odbiega od funkcjonowania opisywanych wcześniej rozwiązań, gdyż należy ono do zupełnie innej kategorii. Zgodnie z wprowadzonym we wcześniejszym rozdziale podziałem nie jest to model dedykowany, lecz rozwiązanie wykorzystujące duże modele językowe wraz z techniką prompt engineering. Oznacza to, że żaden trening nie jest potrzebny, a nacisk przeniesiony został na skonstruowanie wejścia do modelu GPT-3.5-Turbo, które pozwoli jak najlepiej aktywować i wykorzystać zawartą w nim wiedzę. Dane wejściowe do tego modelu mają postać konwersacji, czyli naprzemiennie występujących wiadomości od człowieka oraz od inteligentnego asystenta. Wyjściem jest natomiast wiadomość od asystenta, będąca kontynuacją tej konwersacji.

W celu przewidzenia kompletnego zapytania SQL model GPT-3.5-Turbo jest wykorzystywany w trzech etapach. W pierwszej kolejności przekazywane jest mu pytanie wraz ze schematem bazy danych i proszony jest o zwrócenie nazw tabel posortowanych względem istotności. W drugim kroku w instrukcji wejściowej przekazywane jest pytanie, cztery najistotniejsze tabele wraz z kolumnami oraz relacje między nimi i model proszony jest o posortowanie kolumn w obrębie każdej tabeli od najbardziej do najmniej istotnej. Ostatecznie do GPT-3.5-Turbo przekazywane jest pytanie, elementy schematu uznane za istotne, klucze obce oraz wartości z bazy danych znalezione w procesie schema linking i jest on proszony o wygenerowanie gotowego zapytania.

Podczas tworzenia C3 wykorzystano 3 istotne techniki, od których model wziął swoją nazwę. Określono jest bowiem jako Clear Prompting, Calibration with Hints oraz Consistent Output. Pierwsza z nich polega na tworzeniu instrukcji wejściowych o przejrzystym układzie i zamieszczaniu w nich jedynie elementów najważniejszych. Druga sprowadza się do obserwacji odchyleń względem oczekiwań w odpowiedziach modelu językowego i podawaniu we wcześniejszej konwersacji wskazówek, które to korygują. Ostatnia technika oznacza intensywne wykorzystanie strategii nazywanej self-consistency (X. Wang i in. 2022). Jest ona związana z faktem, że odpowiedzi modeli językowych nie są deterministyczne – przekazywanie tych samych informacji wejściowych zwykle skutkuje różnymi odpowiedziami. W związku z tym często dobrym pomysłem jest podanie do modelu tych samych informacji kilka razy, zebranie wszystkich odpowiedzi i wyłonienie spośród nich tej najczęstszej.

6.4.2 Modyfikacje dla języka polskiego

Z punktu widzenia przystosowania C3 do języka polskiego najistotniejsza była modyfikacja promptów, czyli instrukcji wejściowych do modelu językowego. Poza tym, tak jak w przypadku RAT-SQL, należało zamienić wyrazy stop words z angielskich na polskie. Są one wykorzystywane na etapie schema linking, podczas poszukiwania połączeń value link.

Modyfikacji promptów można dokonać na różne sposoby i rozważano kilka z nich. Pierwszą jest przetłumaczenie w całości na język polski. Wymaga to jednak całkowitej modyfikacji i w związku z tym wysiłek twórców C3 włożony w ich dopracowanie jest w dużym stopniu tracony. Poza tym modele językowe są uczone w większości na danych angielskich i to dla nich osiągają najlepsze wyniki. Drugim z rozważanych podejść był kompletny brak tłumaczenia promptów. Nagłe pojawienie się polskiego pytania oraz elementów schematu w instrukcji wejściowej wydaje się jednak być dość nietypowe, co przypuszczalnie może pogarszać zwracane wyniki. Z wymienionych powodów zdecydowano się pozostawić prompty w języku angielskim, lecz wprowadzić w nich delikatne modyfikacje. W instrukcjach służących do znalezienia istotnych tabel i kolumn słowo question określono przymiotnikiem polish. W instrukcji służącej do finalnego generowania zapytania zrobiono to samo ze słowem databases, przetłumaczono dwa przykładowe pytania na język polski oraz dodano zdanie mówiące, że pytanie zostanie dostarczone w języku polskim. Zmodyfikowane prompty z zaznaczonymi zmianami można znaleźć w dodatku A. Bez wątpienia ciekawym byłoby eksperymentalne zbadanie skuteczności każdego z przedstawionych sposobów modyfikacji instrukcji, lecz byłoby to również kosztowne.

6.4.3 Eksperymenty

Zmodyfikowane w powyżej opisany sposób rozwiązanie C3 chciano przetestować na tych samych zbiorach, co wszystkie wcześniejsze. Problem stanowią w tym przypadku jednak koszty naliczane przez OpenAI za wykorzystanie modelu GPT-3.5-Turbo, które są znaczące. Aby temu sprostać postanowiono skonstruować mniejsze odpowiedniki posiadanych zbiorów. Poszczególne próbki wybrano przy tym tak, aby pod względem rozkładu poziomów trudności zapytań SQL zbiory mniejsze jak najdokładniej odpowiadały oryginalnym. W przypadku zbiorów zawierających elementy schematu zarówno w języku polskim i angielskim zadbano także o to, aby proporcje pomiędzy tymi częściami nie uległy zmianie. Zbiory ze schematem w jednym języku postanowiono zredukować do 50 przykładów, natomiast z dwujęzycznymi schematami do 100 przykładów.

Wykonanie ewaluacji na wybranych pięciuset przykładach spowodowało naliczenie kwoty około 10 dolarów, czyli w przybliżeniu 40 złotych. W przeliczeniu na jedno pytanie daje to około 8 groszy. Względnie wysokie koszty są w dużym stopniu spowodowane wykorzystaniem wspomnianego mechanizmu self-consistency, który wymaga odpytywania modelu językowego wielokrotnie o to samo. Modyfikacja paru parametrów umożliwia zmniejszenie liczby tych odpytań, lecz autorzy metody pokazują w artykule, że wpływa to bezpośrednio na dokładność.

6.4.4 Wyniki

Wyniki wykonanej ewaluacji przedstawiono w tabeli 6.8. Mają one format podobny do wcześniejszych, lecz ze względu na małą liczność zbiorów podawanie części ułamkowej poszczególnych metryk było bezzasadne. W nazwach zbiorów zawarto liczbę znajdujących się w nich przykładów.

Zbiór	Easy	Medium	Hard	Extra Hard	Razem
Pol-Spider-100	54 / 54 / 83	43 / 43 / 91	44 / 38 / 44	13 / 13 / 75	41 / 40 / 79
Pol-Spider-PL-50	58 / 58 / 83	50 / 50 / 91	38 / 38 / 63	13 / 13 / 75	44 / 44 / 82
Pol-Spider-EN-50	50 / 50 / 83	36 / 36 / 91	50 / 38 / 25	13 / 13 / 75	38 / 36 / 76
Pol-Spidersyn-100	32 / 32 / 55	39 / 36 / 61	28 / 27 / 56	0 / 0 / 38	29 / 28 / 55
Pol-Spiderdk-100	100 / 75 / 75	50 / 48 / 59	36 / 36 / 79	15 / 15 / 50	51 / 45 / 63
Pol-Sparc-100	61 / 61 / 79	22 / 18 / 68	0 / 0 / 50	0 / 0 / 0	46 / 45 / 73
Pol-Cosql-100	62 / 56 / 87	31 / 31 / 77	29 / 22 / 36	0 / 0 / 50	44 / 40 / 74

Tabela 6.8: Wyniki modelu C3 na poszczególnych zbiorach. *Wartości w każdej komórce posiadają format EM Without Values / EM / EX i zostały wyrażone w procentach.*

6.4.5 Analiza

Zgodnie z rankingiem zbioru Spider analizowane teraz rozwiązanie jest spośród wszystkich w tej pracy omawianych najlepsze pod kątem metryki EX. Przeprowadzone eksperymenty to potwierdzają. Najlepsze wyniki nie zostały osiągnięte jedynie w przypadku zbioru Pol-Spidersyn, lecz również są dość wysokie.

Wartość metryki EM osiągnęła z drugiej strony zaskakująco niską wartość. Pod jej kątem C3 wydaje się być najgorszym z analizowanych rozwiązań. Skąd wynika tak duża rozbieżność? Powodem jest prawdopodobnie fakt, iż metryka EM dokonuje strukturalnego porównania wzorcowych i przewidzianych zapytań, a ten sam cel może być realizowany przez dwa zapytania całkowicie różne. W przypadku wcześniejszych rozwiązań problem ten nie był tak widoczny, ponieważ były one uczone na części treningowej i w związku z tym nauczyły się bardziej preferować pewne sposoby generowania zapytań od innych. C3 produkuje za to zapytania w sposób znacznie bardziej swobodny, co powoduje problemy ze wstrzeleniem się w oczekiwaną odpowiedź. Nie oznacza to jednak, że jest ona błędna.

Trzeba koniecznie zwrócić uwagę na to, że mimo ogólnie zaniżonej metryki EM, udało się osiągnąć najlepszy dotychczas wynik na zbiorze Pol-Spiderdk, który zawiera wiedzę domenową. C3 przebija dotychczas najlepszy model RAT-SQL o druzgoczące 10 punktów procentowych. Jest to uzasadnione, ponieważ wewnętrznie wykorzystywany model GPT-3.5-Turbo został nauczony wcześniej na ogromnym zbiorze danych i zyskał przez to specjalistyczną wiedzę w bardzo wielu dziedzinach.

Słusznie może pojawić się obawa, że otrzymywane z wykorzystaniem C3 wyniki ewaluacji nie są wiarygodne. Przypuszczalnie model językowy GPT-3.5-Turbo mógł bowiem być trenowany na wykorzystywanych podczas ewaluacji danych lub im podobnych i w związku z tym osiągać w testach zawyżone wyniki. Sytuacja taka nazywana jest wyciekiem danych. Zbiory treningowe nie mogły zawierać przykładów identycznych, gdyż żadne polskie tłumaczenia zbioru Spider wcześniej nie istniały. Artykuł *Rethinking Benchmark and Contamination for Language Models with Rephrased Samples* (S. Yang i in. 2023) wskazuje jednak, że kontaminacja może zostać wprowadzona do zbioru treningowego także poprzez tłumaczenia na inne języki. Z tego powodu angielski zbiór Spider oraz jego istniejące już tłumaczenia mogą stanowić zagrożenie dla rzetelności testów.

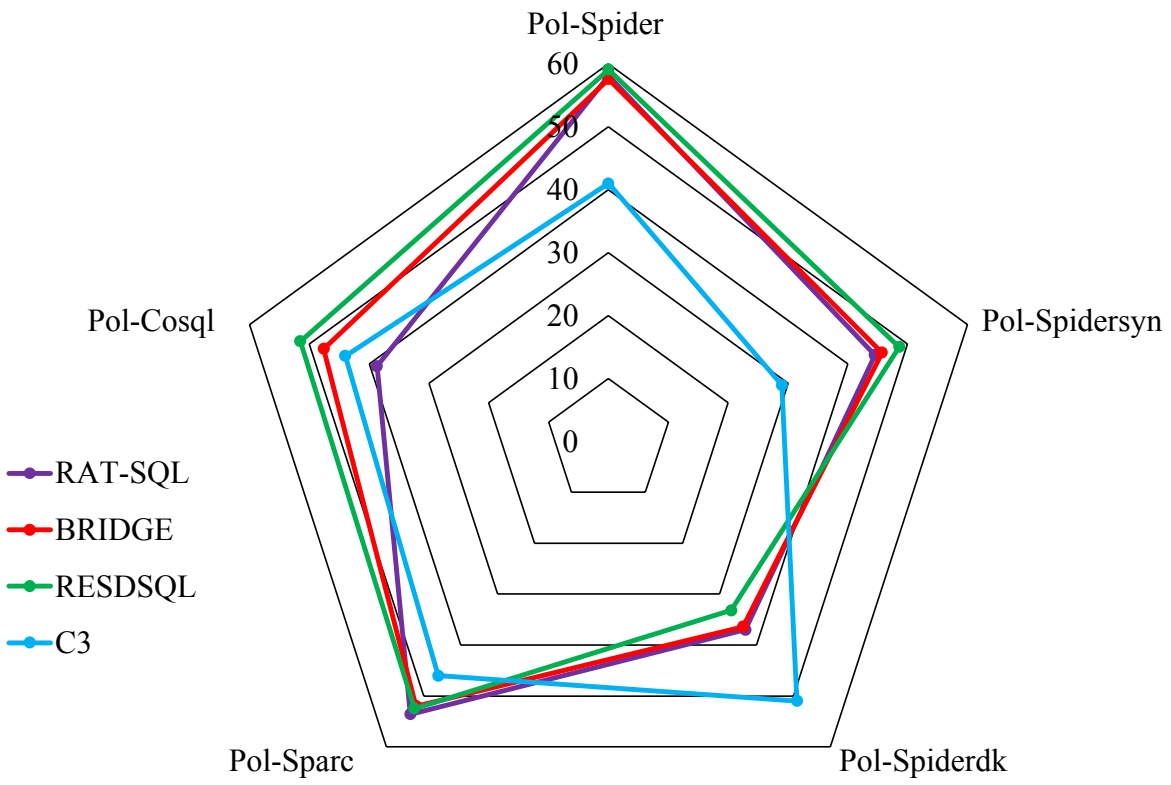
Jeśli chodzi o ranking zbioru Spider, to takiej obawy nie ma, ponieważ część testowa jest utajona i nie ma możliwości, aby OpenAI, nawet przez przypadek, dopuściło do dołączenia tych danych do zbioru treningowego. W niniejszej pracy, jak też wielu innych, z powodu braku dostępności części testowej jej rolę przejmuje publicznie dostępna część pierwotnie nazwana walidacyjną, więc obawa o wyciek danych staje się realna. Zgodnie z artykułem wprowadzającym model GPT-3 (Brown i in. 2020) OpenAI dokonuje usuwania ze zbioru treningowego danych pokrywających się z wykorzystywanymi przez nich zbiorami testowymi. Swojego modelu nie testowali na zbiorze Spider, więc prawdopodobnie specjalne wysiłki nie zostały podjęte w celu wykluczenia tych danych z treningu. Trzeba się więc liczyć z tym, że dane przedstawione w powyższej tabeli mogą być zawyżone.

W przypadku C3 należy zaakcentować dość długi czas potrzebny na wygenerowanie odpowiedzi na dostarczone pytania, ponieważ dla każdego wymagane jest do 30 sekund oczekiwania. Inną zauważoną wadą jest bardzo duża zależność od OpenAI. Wiąże się to oczywiście z kosztami, ale wprowadza również komplikacje w kwestii utrzymania działania całego rozwiązania w dłuższej perspektywie, gdyż OpenAI może dowolnie zmieniać cenniki, usuwać stare modele, czy modyfikować interfejsy. Nie posiada się już kontroli nad całym systemem.

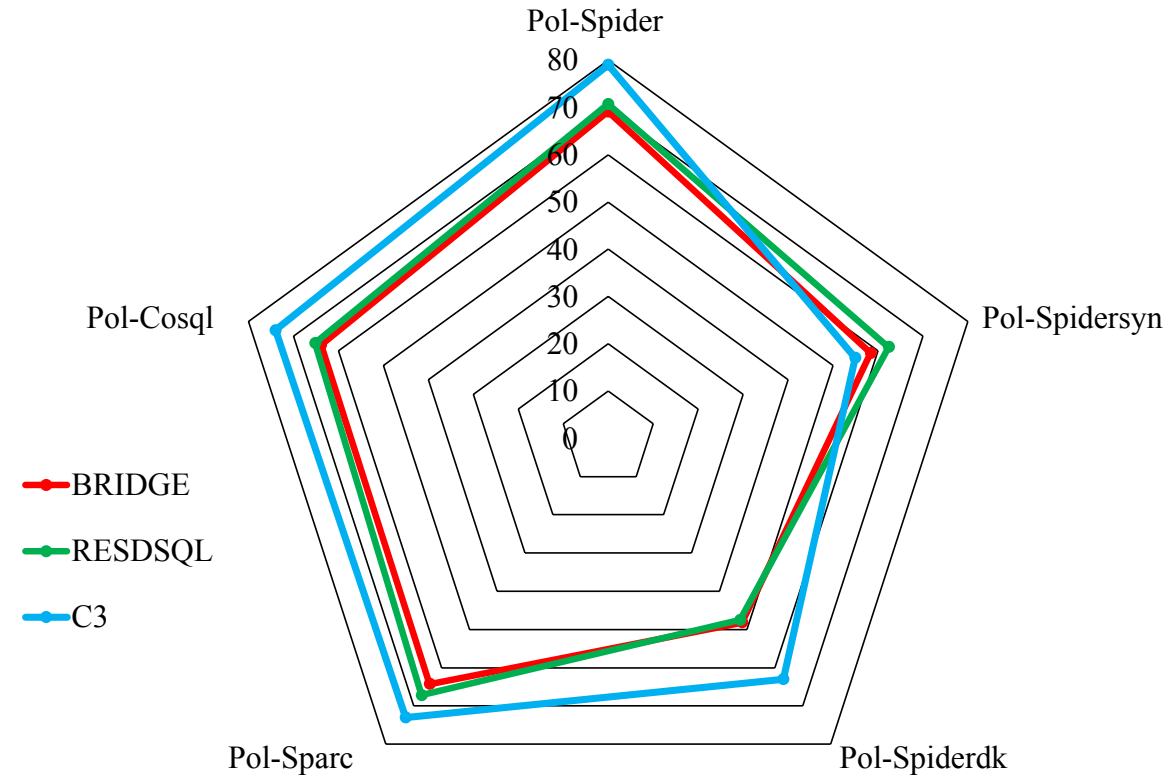
6.5 Podsumowanie wyników

W czterech poprzednich podrozdziałach przedstawione zostały wyniki osiągnięte przez cztery modele. Te rezultaty w tabelarycznej formie, pomimo dostarczania wielu informacji, nie są wygodne do porównywania ze sobą. Zauważono, że wyniki metryk dla wszystkich modeli i zbiorów można wygodnie przedstawić za pomocą wykresów radarowych. Umieszczono je na rysunkach 6.4 oraz 6.5, odpowiednio dla metryki EM Without Values oraz EX.

Zgodnie z wykresem pokazującym wartości metryki EM najlepiej działającym modelem zdaje się być RESDSQL. Zgodnie z drugim wykresem najwyższą skuteczności wykazuje za to bezsprzecznie model C3, a RESDSQL jest na drugiej pozycji. Tak więc te dwa rozwiązania wydają się najbardziej obiecujące i w związku z tym zostaną w kolejnym rozdziale zintegrowane z interfejsem graficznym i poddane dalszym testom.



Rys. 6.4: Wyniki metryki `EM Without Values` dla wszystkich modeli i zbiorów. *Wartości wyrażono procentowo.*



Rys. 6.5: Wyniki metryki `EX` dla wszystkich modeli i zbiorów. *Dla modelu `RAT-SQL` metryki `EX` nie da się obliczyć. Wartości wyrażono procentowo.*

7 Finalne rozwiązanie

Wszystkie przeprowadzone we wcześniejszej części testy modeli sprowadzały się wyłącznie do oceny automatycznej. Wyciągnięto z niej wniosek, że bardzo dobrze spisuje się model RESDSQL, a C3 również jest godny uwagi. Z tego względu w niniejszym rozdziale postanowiono dokonać ponownego treningu RESDSQL, lecz z wykorzystaniem większej wersji modelu T5 oraz większej ilości danych. Tak jak wcześniej dokonano automatycznej oceny, lecz poza tym zaimplementowano interfejs graficzny, pozwalający na praktyczne wykorzystanie oraz manualną ocenę dwóch wybranych modeli. Stworzoną aplikację przedstawiono kilku osobom w celu przetestowania i wyrażenia swojej opinii. Te skrótowo opisane działania zostaną w kolejnych sekcjach rozwinięte.

7.1 Ponowny trening RESDSQL

Jako że wyniki modelu RESDSQL okazały się najlepsze spośród wszystkich klasycznych, trenowalnych rozwiązań, postanowiono dokonać jego nauki ponownie, lecz z wykorzystaniem rozszerzonego zbioru oraz większego wariantu modelu T5, licząc na jeszcze lepsze rezultaty.

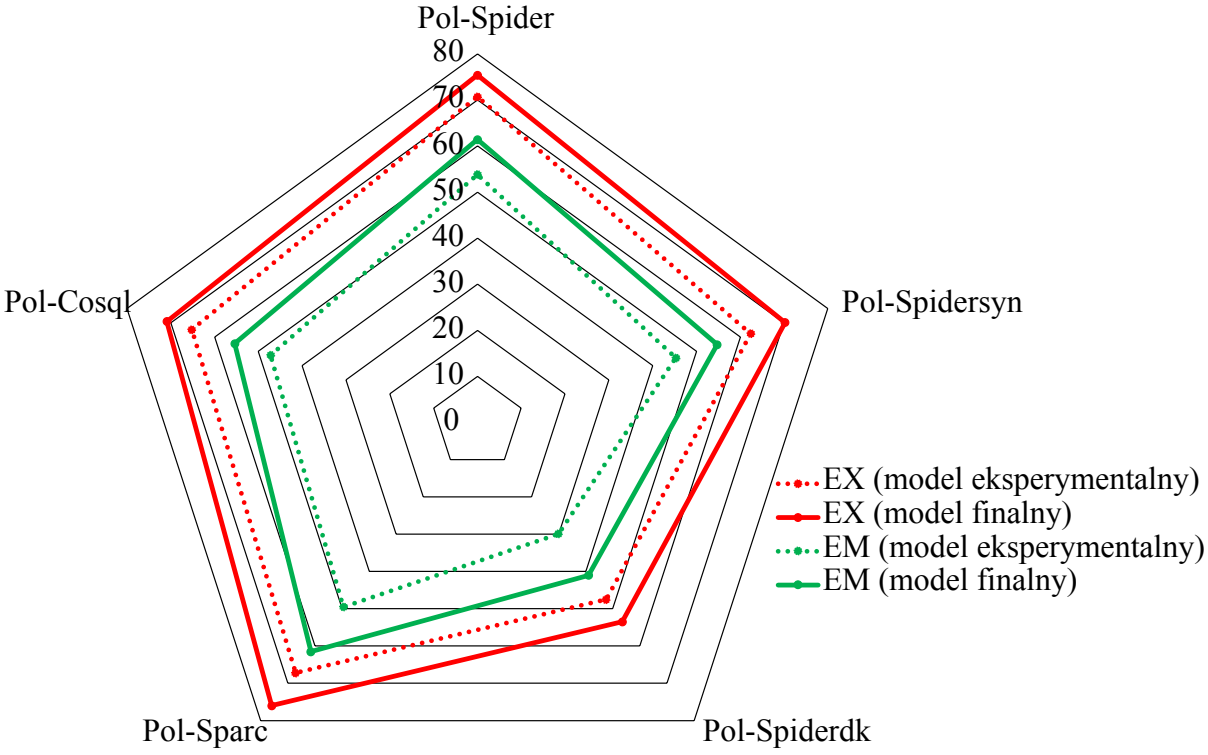
Wcześniej do treningu wykorzystano jedynie warianty zbioru Spider, ponieważ pozwoliło to skrócić czas treningu oraz wykonywać rzetelne porównania z modelami tworzonymi przez innych badaczy, którzy również dokonywali treningu wyłącznie na tym zbiorze. Teraz poza zbiorem Spider do danych treningowych postanowiono włączyć Spider-Syn, SParC oraz CoSQL w wariantach z polskimi i angielskimi schematami baz danych, a także ich angielskie pierwowzory. Zbiór Spider-DK pominięto, ponieważ zawiera on jedynie przykłady testowe. Posługując się wcześniej wprowadzonymi nazwami, można powiedzieć inaczej, że do zbioru treningowego wybrano Mix-Spider, Mix-Spidersyn, Mix-Sparc oraz Mix-Cosql. W ten sposób ilość danych treningowych została zwiększona prawie dwukrotnie.

Wariant modelu T5 postanowiono zmienić z wcześniej wykorzystanego mt5-base na mt5-large, który zawiera dwukrotnie większą liczbę parametrów. Powoduje ona, że model jest w stanie pomieścić znacznie większą wiedzę. W fazie eksperymentów nie skorzystano z tego, ponieważ nie dysponowano kartą graficzną o odpowiednio dużej ilości pamięci VRAM. **Stało się to możliwe dzięki uprzejmości p. Grzegorza Warzechy z firmy TaskPilot, który udostępnił odpowiednie zasoby sprzętowe** i do treningu wykorzystano potężną kartę Nvidia RTX 4090, dysponującą 24 GB pamięci VRAM.

Na rozwiązanie RESDSQL składają się dwa modele. Pierwszego, oceniającego istotność tabel i kolumn, nie uczono ponownie, lecz wykorzystano ten wytrenowany wcześniej. Zamiana rozmiaru modelu językowego T5 i ponowny trening tyczy się jedynie drugiego etapu, w którym następuje ostateczne generowanie zapytań SQL. Naukę kontynuowano przez 14 godzin, a wyniki kompletnego modelu przedstawiono w tabeli 7.1. Porównanie wyników osiąganych przez ten finalny model z modelem eksperymentalnym przedstawiono natomiast na rysunku 7.1.

Zbiór	Easy	Medium	Hard	Extra Hard	Razem
Pol-Spider	85,5/76,8/87,7	73,4/66,6/78,8	52,6/48,3/66,7	41,6/38,0/57,2	67,7/61,4/75,4
Pol-Spider-PL	86,7/78,6/87,9	74,7/67,9/79,4	54,0/48,9/67,8	41,0/36,7/56,6	68,7/62,3/75,8
Pol-Spider-EN	84,3/75,0/87,5	72,2/65,2/78,3	51,1/47,7/65,5	42,2/39,2/57,8	66,7/60,4/75,0
En-Spider	86,7/83,5/88,3	76,9/73,8/79,6	59,2/56,9/65,5	47,6/47,0/53,0	71,6/69,0/75,0
Pol-Spidersyn	72,2/64,0/80,4	64,3/60,3/75,4	46,0/45,6/60,7	38,0/35,5/51,7	58,7/54,7/70,2
Pol-Spiderdk	63,2/58,2/67,7	49,2/44,1/56,1	33,1/33,1/48,6	22,9/21,4/35,7	44,7/41,0/53,5
Pol-Sparc	77,5/72,1/82,7	50,0/44,2/66,0	20,0/20,0/50,0	43,8/43,8/50,0	66,7/61,5/76,0
Pol-Cosql	73,8/67,9/80,0	61,0/54,2/68,6	27,9/27,9/52,9	29,4/26,5/50,0	60,4/55,4/70,9

Tabela 7.1: Wyniki finalnego modelu RESDSQL na poszczególnych zbiorach. *Wartości w każdej komórce posiadają format EM Without Values / EM / EX i zostały wyrażone w procentach.*



Rys. 7.1: Porównanie wyników eksperymentalnego i finalnego modelu RESDSQL. *Metryki wyrażono w procentach.*

Wyniki testów pokazują, że zwiększenie rozmiaru modelu T5, wraz z rozszerzeniem zbioru treningowego, pozwoliło wyraźnie podnieść wyniki wszystkich metryk w porównaniu do wcześniej trenowanej wersji. W przypadku każdego ze zbiorów testowych nastąpiło zwiększenie wartości metryki EM o przynajmniej 8 punktów procentowych. Wartość metryki EX również wzrosła, co sprawiło, że ta wersja modelu RESDSQL dorównuje pod tym kątem prawie modelowi C3.

7.2 Aplikacja webowa

W celu umożliwienia praktycznego wykorzystania oraz manualnej weryfikacji modeli RESDSQL oraz C3 opracowany został interfejs graficzny w postaci aplikacji webowej. W szczególności duże wyzwanie stanowiło połączenie go z ze wspomnianymi modelami, ponieważ zostały stworzone do wykonywania predykcji na określonych zbiorach w sposób wsadowy. W tym przypadku musiały być jednak wykonywane dla pojedynczych pytań oraz dowolnych baz danych.

7.2.1 Interfejs graficzny

Interfejs graficzny, którego wygląd przedstawiono na rysunkach w dodatku B, został stworzony z wykorzystaniem biblioteki streamlit. Jak opisano w rozdziale o narzędziach, jej zaletą jest niebywała prostota, lecz wiąże się ona z pewnym brakiem elastyczności – wpływanie na wygląd poszczególnych komponentów oraz interakcje pomiędzy nimi jest bardzo ograniczone. Przygotowana aplikacja wydaje się dość skomplikowana jak na możliwości tej biblioteki, lecz po dokładnym przemyśleniu struktury i szczegółowym zapoznaniem z dokumentacją, udało się ją zrealizować. Na aplikację składają się trzy zakładki: pierwsza pozwala na wybór bazy danych, druga na podanie naturalnych odpowiedników dla nazw tabel i kolumn, a trzecia na generowanie zapytań SQL.

Pierwsza zakładka umożliwia wybór bazy danych poprzez załadowanie jej w formacie sqlite, wybranie jednej z przygotowanych baz przykładowych lub wprowadzenie skryptu SQL, który ją tworzy. Sześć przykładowych baz wybrano z części testowej zbioru Spider, lecz dodatkowo dokonano tłumaczenia ich zawartości. Po dokonaniu decyzji ukazuje się diagram wybranej bazy, do zaimplementowania którego wykorzystano bibliotekę języka Python o nazwie eralchemy[13]. Pozwala ona w łatwy sposób tworzyć diagramy relacji encji (ang. entity relation diagram). Dzięki

[13] https://pypi.org/project/ERAlchemy

temu możliwe jest powierzchowne zapoznanie się ze strukturą bazy i uświadomienie pytań, które można podawać do obsługującego ją interfejsu tekstowego.

Druga zakładka pozwala na podanie naturalnych odpowiedników dla nazw tabel i kolumn. Jej obecność jest wymuszona przez strukturę zbioru `Spider`, który zawiera taką informację oraz fakt, że rozwiązania `RESDSQL` oraz `C3` z niej korzystają. Jest to pewnego rodzaju informacja dodatkowa, której podawanie może być irytujące, szczególnie w przypadku dużych baz danych. Aby to usprawnić, opracowano prosty algorytm, który wstępnie proponuje nazwy naturalne poprzez rozbicie identyfikatorów tabel i kolumn na poszczególne słowa. W dużej mierze ogranicza to działania użytkownika do ich weryfikacji.

Trzecia zakładka umożliwia zadawanie pytań do określonej wcześniej bazy danych. Ma to postać chatu, w którym naprzemiennie użytkownik podaje swoje żądania oraz model wysyła przewidziane zapytania SQL. Forma ta może być nieco myląca, ponieważ sugeruje, że wykorzystywane modele działają w sposób kontekstowy, czyli biorą pod uwagę wcześniejszą historię wiadomości, podczas gdy tak nie jest. Jest ona jednak wygodna, gdyż większość użytkowników jest z nią zaznajomiona. Generowane zapytania są natychmiastowo wykonywane i interaktywne tabele z odczytanymi rekordami są bezpośrednio pod nimi zamieszczane, co jest bardzo ważne pod kątem praktycznego wykorzystania. Aktywny w danej chwili model można zmieniać za pomocą listy rozwijanej. Jeżeli wybrany został `C3`, to konieczny jest jednak dostęp do klucza API `OpenAI`, który można wprowadzić w odpowiednim polu tekstowym w interfejsie. Przewidziano również możliwość uruchomienia całej aplikacji z danym kluczem, a wówczas podawanie go przez poszczególnych użytkowników nie jest konieczne. Dostęp do interfejsu chroniony jest wówczas hasłem, by przeciwdziałać naliczaniu kosztów przez nieuprawnione osoby.

7.2.2 Połączenie interfejsu z modelami

Dość kłopotliwe okazało się połączenie stworzonego interfejsu z modelami, w szczególności z `RESDSQL`. W przypadku obu rozwiązań istniejący kod nie pozwala bowiem na proste generowanie zapytań do dowolnych baz danych. Wymagają, aby przykłady testowe były dostarczane w formacie wykorzystywanym przez zbiór `Spider`. Z tego powodu posiadając wybraną przez użytkownika bazę oraz naturalne odpowiedniki nazw tabel i kolumn, konieczne okazało się każdorazowe generowanie tymczasowego, jednoelementowego zbioru danych, co wymagało dodatkowego wysiłku.

Innego rodzaju problemem było generowanie przez RESDSQL zapytań w sposób, który można nazwać wsadowym. Odbywa się ono bowiem w dwóch etapach, z których każdy korzysta z osobnej sieci neuronowej. Istniejący kod działa w ten sposób, że ładuje do pamięci model pierwszy i przepuszcza przez niego wszystkie dane, a następnie zastępuje go drugim i podaje do niego wszystkie wyniki pośrednie. Sprawdza się to w przypadku generowania dużej liczby zapytań SQL jednocześnie, lecz niekoniecznie dla serii pojedynczych zapytań, tak jak w rozpatrywanym przypadku. Okazuje się wówczas, że większość czasu poświęcana jest naprzemiennemu ładowaniu sieci neuronowych do pamięci zamiast rzeczywistym obliczeniom. Z tego powodu zmodyfikowano kod w taki sposób, aby w pamięci znajdywały się przez cały czas obie sieci. Oznacza to oczywiście wykorzystanie znacznie większej ilości pamięci, czemu postanowiono przeciwdziałać poprzez załadowanie parametrów modeli z wykorzystaniem precyzji 16-bitowej zamiast oryginalnie 32-bitowej. Dzięki temu wymaganie pamięciowe zostało zredukowane do 10 GB pamięci VRAM, czy też RAM w przypadku braku odpowiedniej karty graficznej, lecz zwiększa to czas generowania zapytań z kilku do kilkunastu sekund.

7.3 Manualna ocena

W celu dokonania manualnej oceny modeli RESDSQL i C3, a także całej aplikacji, dokonane zostały przez autora pracy dogłębne testy. Poza tym wybrana została grupa 5 studentów, z których każdy poświęcił około 30 minut na zapoznanie się z aplikacją i wyraził na jej temat opinię. Czworo z nich studiowało nauki techniczne i znało język SQL, lecz w bardzo różnym stopniu. W badaniu uczestniczyła również studentka psychologii, która o istnieniu języka SQL nawet nie miała pojęcia. Od tych chętnych do przetestowania aplikacji osób starano się uzyskać między innymi opinię na temat wygody użytkowania oraz każdego z dostępnych modeli. Pytano również, czy znaleźliby u siebie dla niej zastosowanie i aktywnie korzystali.

7.3.1 Wygoda korzystania

Błahym problemem, który nie spotkał większości osób, były sporadyczne awarie aplikacji – przestawała ona reagować. Każdy wykonywał testy na własnym komputerze, gdyż aplikacja została tymczasowo udostępniona w sieci i wystarczyło wejść na wskazany adres. Z pewnością pojawienie się tej niedogodności jest uwarunkowane środowiskiem, w którym aplikacja jest używana. Autorowi nie udało się niestety odtworzyć tego błędu na własnym sprzęcie, nawet

korzystając z tych samych przeglądarek, co dwie osoby, które go napotkały. Tak więc problem pozostał, lecz nie uznano jego usunięcia za kluczowe z punktu widzenia realizowanego tematu.

Inną niedogodnością, którą każda z testujących osób w mniejszym lub większym stopniu podkreśliła, jest konieczność wprowadzania naturalnych odpowiedników dla nazw schematu. W przypadku baz danych, gdzie schemat jest w języku angielskim, nie stanowi to wielkiego problemu, ponieważ zaimplementowany mechanizm skutecznie rozdziela kilkuelementowe identyfikatory na słowa. W przypadku polskich schematów ochotnicy musieli jednak ręcznie dodawać polskie znaki, co niektórych wyraźnie frustrowało. Jest to na pewno uzasadnione, gdy rozważana aplikacja ma być rozwiązaniem inteligentnym, a zmusza użytkowników do robienia tak przyziemnych rzeczy, jak dodawanie polskich znaków. Z pewnością można w przyszłości tę część usprawnić.

7.3.2 Dokładność modeli

Żadna z osób nie miała większych problemów z zadawaniem pierwszych pytań. Często jednak były one szczegółowe, a przykładowe bazy danych nie zawierają wiele informacji, więc zapytania SQL nie zwracały żadnych rekordów. Sugerowano wówczas, aby rozpocząć od zadawania pytań bardziej ogólnych, które potem stawały się coraz bardziej skomplikowane. Na początku większość osób badała model `RESDSQL`, gdyż działając na `GPU` zwraca odpowiedzi niemal natychmiastowo. Gdy zauważyli błąd w zapytaniu, to wówczas ponawiali pytanie, lecz z modelem `C3`, który niemal zawsze radził sobie lepiej. Ostatecznie każdy z uczestników badania wypracował sobie opinię, że model `C3` jest dokładniejszy. Pytani o to, czy dłuższy czas czekania im nie przeszkadza, stwierdzali, że kilkanaście sekund nie stanowi problemu. Jedna osoba zauważyła, że niemal każdy korzysta już z asystenta `ChatGPT` (OpenAI 2023) i przyzwyczailiśmy się do pewnych opóźnień.

Zauważono kilka scenariuszy, w których szczególnie widać wyższość modelu `C3`. Jednym z nich jest pytanie o konkretne obiekty z bazy danych, gdyż `C3` potrafi lepiej odmieniać różne wyrażenia. Przykładowo, gdy zapytamy o wszystkie psy należące do Sary, to wysoce prawdopodobne, że `RESDSQL` zwróci oczywiście błędne zapytanie z fragmentem `WHERE imie="Sary"`, a `C3` zadziała poprawnie. Innym scenariuszem jest zadawanie nietypowych dla języka SQL zapytań, na przykład „Podaj dla każdego właściciela listę psów” lub „Zwróć imiona psów, które zaczynają się na tę samą literę, na którą kończy się imię właściciela”. Wymagają one znajomości funkcji `group_concat` oraz `substr`, których w zbiorze treningowym `RESDSQL` się nie doświadczy. `C3` nie posiada

natomiast w tej kwestii żadnych ograniczeń, ma bowiem dostęp do ogromnej wiedzy modelu GPT-3.5-turbo, trenowanego na danych z sieci.

Obserwując zadawane przez testerów pytania i zwracane przez modele odpowiedzi, zauważono, że często zapytania produkowane przez RESDSQL można uznać za poprawne, lecz nie do końca zgodne z tym, czego spodziewali się użytkownicy. Przykładowo zadając pytanie „Zwróć liczbę studentów na każdym semestrze" użytkownicy spodziewają się zobaczyć tabelę z nazwami semestrów oraz liczbą studentów na każdym z nich. W przypadku takiego zapytania model może natomiast zwrócić wyłącznie liczby studentów. Ewentualnie doda semestry, lecz w postaci ich liczbowych identyfikatorów. W pytaniu nie było określone, że te nazwy semestrów mają się pojawić. Jest to jednak wiedza domenowa, z którą model C3 radzi sobie lepiej, co wykazały nawet wcześniejsze testy na zbiorze Pol-Spiderdk.

Dokładność modeli można podsumować twierdzeniem, że model C3 produkuje w praktyce wyraźnie lepsze zapytania względem RESDSQL. Odbywa się to kosztem dłuższego czasu oczekiwania, lecz nie stanowi to dużego problemu. Mimo wszystko C3 nie jest bezbłędny i zdarza mu się pomylić nazwy kolumn, błędnie dokonać złączenia tabel, czy niepoprawnie zrozumieć żądanie użytkownika.

7.3.3 Praktyczne zastosowanie

Większość pytanych osób stwierdziła, że mogłaby znaleźć dla stworzonego rozwiązania zastosowanie, chociażby w swojej pracy zawodowej. Uzasadniały, że modele nie zawsze produkują prawidłowe zapytania, ale ich poprawienie wymaga mniej wysiłku. Negatywną opinię wyraziła jednak osobą z najlepszą znajomością języka SQL, bo uznała, że z testowanej aplikacji by nie korzystała. Uważa ona, że więcej czasu potrzebuje na wygenerowanie i weryfikację zapytania, niż napisanie go od zera. Wskazała jednak na użyteczność dla mniej doświadczonych osób.

Konieczne jest niestety podkreślenie, że ze stworzonej aplikacji nie powinien korzystać każdy. Zaobserwowano to na przykładzie testującej aplikację osoby, która SQL nie znała. Zdarzało się, że produkowane zapytania były wykonywalne i zwracały wyniki, wydające się być poprawnymi, lecz było inaczej. Osoba ta nie mogła zweryfikować poprawności zapytania SQL, co może być wręcz niebezpieczne i prowadzić do wyciągania nieprawidłowych wniosków. Tak więc opracowane rozwiązanie nie realizuje szczytnego celu, jakim było zapewnienie każdemu łatwego dostępu do informacji, ponieważ minimalna znajomość języka SQL wciąż jest koniecznością.

8 Podsumowanie

W ramach realizacji ważnego i aktualnego tematu, jakim jest tłumaczenie zapytań z języka polskiego na SQL, najpierw dokonano przeglądu istniejących jedynie dla innych języków zbiorów danych. Przeanalizowano różnice w podejściach ich autorów i wybrano optymalne dla przypadku języka polskiego. Następnie poświęcono dużo uwagi na przygotowanie odpowiednich zbiorów i z ich wykorzystaniem przetestowano wybrane rozwiązania. Okazało się, że najlepsze rezultaty w fazie eksperymentów osiągnęły znacznie różniące się od siebie modele `RESDSQL` oraz `C3`, z których pierwszy został poddany powtórnemu, rozszerzonemu treningowi. Modele te zostały zintegrowane z aplikacją internetową umożliwiającą interaktywne generowanie i wykonywanie zapytań SQL. Podczas manualnej oceny użytkownicy dobrze ocenili działanie systemu, doceniając jego praktyczne zastosowanie. Okazało się jednak, że minimalna znajomość języka SQL wciąż jest potrzebna w celu weryfikacji produkowanych zapytań.

Podsumowując, w ramach pracy zrealizowano wszystkie założone cele, których głównymi punktami było przygotowanie polskich zbiorów dla problemu `Text-to-SQL` i przeprowadzenie z ich wykorzystaniem eksperymentów. Przetestowanie czterech różnych rozwiązań pozwoliło zapoznać się ze sposobem ewolucji wykorzystywanych do tego celu modeli. W trakcie eksperymentów potwierdzono zaobserwowane już wcześniej zjawisko polegające na tym, że modele osiągają lepsze rezultaty, gdy uczone są na dwujęzycznych zbiorach. Pokazano także, że rozwiązania bazujące na zamkniętych modelach językowych, na przykład udostępnianych przez `OpenAI`, mogą osiągać bardzo dobre wyniki, lecz ze względu na swoją specyfikę nie wszędzie się sprawdzą.

W celu ewentualnej kontynuacji niniejszego tematu zdecydowanie warto dokonać ręcznej weryfikacji i poprawek stworzonych z wykorzystaniem tłumaczenia maszynowego zbiorów. Obecność w nich wielu niedociągnięć jest bowiem wiadoma, a ich usunięcie pozwoli trenować wyższej jakości modele. Przypuszcza się, że poprawę skuteczności można uzyskać również poprzez naukę na połączeniu wszystkich istniejących tłumaczeń zbioru `Spider`, zamiast jedynie polskiego z angielskim, co testowali już autorzy `MultiSpider` (Dou i in. 2023). Aby zmniejszyć wymagania pamięciowe modelu `RESDSQL` można spróbować dokonać kwantyzacji 8-bitowej (Noune i in. 2022; Perez i in. 2023). Jeszcze innym pomysłem jest przetestowanie dla polskiego przypadku rozwiązań z aktualnego szczytu rankingu `Spider`, które bazują na udostępnianym przez `OpenAI` modelu `GPT-4`, co w tej pracy pominięto ze względu na koszty.

Bibliografia

Amazon (2024). *Amazon Translate*. URL: https://docs.aws.amazon.com/translate/ (term. wiz. 13. 01. 2024).

Argos Open Tech (2020). *argosopentech/argos-translate: Open-source offline translation library written in Python*. URL: https://github.com/argosopentech/argos-translate (term. wiz. 18. 01. 2024).

Bahasa, Jurnal i Tira Nur Fitria (2023). „Performance of Google Translate, Microsoft Translator, and DeepL Translator: Error Analysis of Translation Result". W: *Analysis of Translation Result. Al-Lisan: Jurnal Bahasa (e-Journal)* 8 (2), s. 115. ISSN: 2442-8973. DOI: 10.30603/al.v8i2.3442.

Bakshandaeva, Daria i in. (2022). „PAUQ: Text-to-SQL in Russian". W: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, s. 2355–2376. URL: https://aclanthology.org/2022.findings-emnlp.175.

Bogoychev, Nikolay, Jelmer Van der Linde i Kenneth Heafield (2021). „TranslateLocally: Blazing-fast translation running on the local CPU". W: arXiv: 2109.10194 [cs.CL].

Brown, Tom i in. (2020). „Language Models are Few-Shot Learners". W: *Advances in Neural Information Processing Systems*. Red. H. Larochelle i in. T. 33. Curran Associates, Inc., s. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Chang, Shuaichen i in. (2023). „Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness". W: arXiv: 2301.08881 [cs.CL].

Church, Kenneth Ward (2017). „Word2Vec". W: *Natural Language Engineering* 23.1, s. 155–162.

Dadas, Sławomir (2019). *A repository of Polish NLP resources*. Github. URL: https://github.com/sdadas/polish-nlp-resources/.

DeepL (2024). *Tłumacz DeepL – najlepszy translator na świecie*. URL: https://www.deepl.com (term. wiz. 13. 01. 2024).

Devlin, Jacob i in. (2019). „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". W: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Red. Jill Burstein, Christy Doran i Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, s. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

Dong, Xuemei i in. (2023a). *bigbigwatermalon/C3SQL: The code for the paper C3: Zero-shot Text-to-SQL with ChatGPT*. URL: https://github.com/bigbigwatermalon/C3SQL (term. wiz. 29. 12. 2023).

— (2023b). „C3: Zero-shot Text-to-SQL with ChatGPT". W: arXiv: 2307.07306 [cs.CL].

Dou, Longxu i in. (2023). „MultiSpider: Towards Benchmarking Multilingual Text-to-SQL Semantic Parsing". W: *AAAI Conference on Artificial Intelligence*. URL: https://ojs.aaai.org/index.php/AAAI/article/view/26499/26271.

Gan, Yujian, Xinyun Chen, Qiuping Huang i in. (2021). „Towards Robustness of Text-to-SQL Models against Synonym Substitution". W: Online: Association for Computational Linguistics, s. 2505–2515. DOI: 10.18653/v1/2021.acl-long.195. URL: https://aclanthology.org/2021.acl-long.195.

Gan, Yujian, Xinyun Chen i Matthew Purver (2021). „Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization". W: arXiv: 2109.05157 [cs.CL].

Gan, Yujian, Xinyun Chen, Jinxia Xie i in. (2021). „Natural SQL: Making SQL Easier to Infer from Natural Language Specifications". W: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, s. 2030–2042. DOI: 10.18653/v1/2021.findings-emnlp.174. URL: https://aclanthology.org/2021.findings-emnlp.174.

Google (2024). *Google Cloud Translation – Automate with AI-powered translation*. URL: https://cloud.google.com/translate (term. wiz. 13. 01. 2024).

Gunasekar, Suriya i in. (2023). „Textbooks Are All You Need". W: arXiv: 2306.11644 [cs.CL].

Guo, Jiaqi i in. (2019). „Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation". W: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Red. Anna Korhonen, David Traum i Lluís Màrquez. Florence,

Italy: Association for Computational Linguistics, s. 4524–4535. DOI: 10.18653/v1/P19-1444. URL: https://aclanthology.org/P19-1444.

Hawkins, Douglas M. (2004). „The Problem of Overfitting". W: *Journal of Chemical Information and Computer Sciences* 44.1, s. 1–12. DOI: 10.1021/ci0342472. eprint: https://doi.org/10.1021/ci0342472. URL: https://doi.org/10.1021/ci0342472.

Hidalgo-Ternero, Carlos Manuel (2021). „Google Translate vs. DeepL: Analysing neural machine translation performance under the challenge of phraseological variation". W: *Monografias de Traduccion e Interpretacion (MonTI)* (Especial 6), s. 154–177. ISSN: 19899335. DOI: 10.6035/MONTI.2020.NE6.5.

José, Marcelo Archanjo i Fabio Gagliardi Cozman (2021). „mRAT-SQL+GAP: A Portuguese Text-to-SQL Transformer". W: t. 13074 LNAI. Springer Science i Business Media Deutschland GmbH, s. 511–525. ISBN: 9783030916985. DOI: 10.1007/978-3-030-91699-2_35.

Katsogiannis-Meimarakis, George i Georgia Koutrika (2021). „A Deep Dive into Deep Learning Approaches for Text-to-SQL Systems". W: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. Virtual Event, China: Association for Computing Machinery, s. 2846–2851. ISBN: 9781450383431. DOI: 10.1145/3448016.3457543. URL: https://doi.org/10.1145/3448016.3457543.

— (sty. 2023). „A survey on deep learning approaches for text-to-SQL". W: *The VLDB Journal* 32.4, s. 905–936. ISSN: 1066-8888. DOI: 10.1007/s00778-022-00776-8. URL: https://doi.org/10.1007/s00778-022-00776-8.

Klein, Guillaume, François Hernandez i in. (2020). „The OpenNMT Neural Machine Translation Toolkit: 2020 Edition". W: *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*. Red. Michael Denkowski i Christian Federmann. Virtual: Association for Machine Translation in the Americas, s. 102–109. URL: https://aclanthology.org/2020.amta-research.9.

Klein, Guillaume, Yoon Kim, Yuntian Deng, Vincent Nguyen i in. (2018). „OpenNMT: Neural Machine Translation Toolkit". W: arXiv: 1805.11462 [cs.CL].

Klein, Guillaume, Yoon Kim, Yuntian Deng, Jean Senellart i in. (2017). „OpenNMT: Open-Source Toolkit for Neural Machine Translation". W: *Proceedings of ACL 2017, System*

Demonstrations. Red. Mohit Bansal i Heng Ji. Vancouver, Canada: Association for Computational Linguistics, s. 67–72. URL: https://aclanthology.org/P17-4012.

Li, Haoyang i in. (2023a). „RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL". W: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'23/IAAI'23/EAAI'23. AAAI Press. ISBN: 978-1-57735-880-0. DOI: 10.1609/aaai.v37i11.26535. URL: https://doi.org/10.1609/aaai.v37i11.26535.

— (2023b). *RUCKBReasoning/RESDSQL: The Pytorch implementation of RESDSQL*. URL: https://github.com/RUCKBReasoning/RESDSQL (term. wiz. 16. 12. 2023).

Li, Jinyang i in. (maj 2023). „Graphix-T5: Mixing Pre-trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing". W: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.11, s. 13076–13084. DOI: 10.1609/aaai.v37i11.26536. URL: https://ojs.aaai.org/index.php/AAAI/article/view/26536.

Lin, Xi Victoria, Richard Socher i Caiming Xiong (2020a). „Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing". W: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Red. Trevor Cohn, Yulan He i Yang Liu. Online: Association for Computational Linguistics, s. 4870–4888. DOI: 10.18653/v1/2020.findings-emnlp.438. URL: https://aclanthology.org/2020.findings-emnlp.438.

— (2020b). *Translating natural language questions to a structured query language*. URL: https://github.com/salesforce/TabularSemanticParsing (term. wiz. 16. 12. 2023).

Ling, Charles X., Jin Huang i Harry Zhang (2003). „AUC: A Better Measure than Accuracy in Comparing Learning Algorithms". W: *Advances in Artificial Intelligence*. Red. Yang Xiang i Brahim Chaib-draa. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 329–341. ISBN: 978-3-540-44886-0.

Liu, Xing, Huiqin Chen i Wangui Xia (2022). „Overview of named entity recognition". W: *Journal of Contemporary Educational Research* 6.5, s. 65–68.

Ma, Jianqiang i in. (2020). „Mention Extraction and Linking for SQL Query Generation". W: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Proces-*

sing (EMNLP). Red. Bonnie Webber i in. Online: Association for Computational Linguistics, s. 6936–6942. DOI: 10.18653/v1/2020.emnlp-main.563. URL: https://aclanthology.org/2020.emnlp-main.563.

Microsoft (2024). *Tłumacz platformy Azure AI | Microsoft Azure*. URL: https://azure.microsoft.com/pl-pl/products/ai-services/ai-translator (term. wiz. 13. 01. 2024).

Min, Qingkai, Yuefeng Shi i Yue Zhang (2019). „A Pilot Study for Chinese SQL Semantic Parsing". W: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, s. 3643–3649.

Mitra, Arindam i in. (2023). „Orca 2: Teaching Small Language Models How to Reason". W: arXiv: 2311.11045 [cs.AI].

Nguyen, Anh Tuan, Mai Hoang Dao i Dat Quoc Nguyen (2020). „A Pilot Study of Text-to-SQL Semantic Parsing for Vietnamese". W: *Findings of the Association for Computational Linguistics: EMNLP 2020*, s. 4079–4085.

Noune, Badreddine i in. (2022). „8-bit Numerical Formats for Deep Neural Networks". W: arXiv: 2206.02915 [cs.LG].

OpenAI (2023). *ChatGPT*. URL: https://chat.openai.com (term. wiz. 21. 01. 2024).

Pennington, Jeffrey, Richard Socher i Christopher Manning (2014). „GloVe: Global Vectors for Word Representation". W: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Red. Alessandro Moschitti, Bo Pang i Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, s. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

Perez, Sergio P. i in. (2023). „Training and inference of large language models using 8-bit floating point". W: arXiv: 2309.17224 [cs.LG].

Quamar, Abdul i in. (2022). „Natural language interfaces to data". W: *Foundations and Trends in Databases* 11.4, s. 319–414. ISSN: 1931-7883. DOI: 10.1561/1900000078.

Radford, Alec i Karthik Narasimhan (2018). „Improving Language Understanding by Generative Pre-Training". W: URL: https://api.semanticscholar.org/CorpusID:49313245.

Raffel, Colin i in. (2020). „Exploring the limits of transfer learning with a unified text-to-text transformer". W: *J. Mach. Learn. Res.* 21.1. ISSN: 1532-4435.

Shi, Peng i in. (2022). „XRICL: Cross-lingual Retrieval-Augmented In-Context Learning for Cross-lingual Text-to-SQL Semantic Parsing". W: *Findings of the Association for Computational Linguistics: EMNLP 2022*. Red. Yoav Goldberg, Zornitsa Kozareva i Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, s. 5248–5259. DOI: 10.18653/v1/2022.findings-emnlp.384. URL: https://aclanthology.org/2022.findings-emnlp.384.

Stack Overflow (2023). *Stack Overflow 2023 Developer Survey*. URL: https://survey.stackoverflow.co/2023/ (term. wiz. 25. 11. 2023).

Vaswani, Ashish i in. (2017). „Attention is All you Need". W: *Advances in Neural Information Processing Systems*. Red. I. Guyon i in. T. 30. Curran Associates, Inc.

Vinyals, Oriol, Meire Fortunato i Navdeep Jaitly (2015). „Pointer Networks". W: *Advances in Neural Information Processing Systems*. Red. C. Cortes i in. T. 28. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Wang, Bailin i in. (2020a). *microsoft/rat-sql: A relation-aware semantic parsing model from English to SQL*. URL: https://github.com/microsoft/rat-sql (term. wiz. 05. 12. 2023).

— (2020b). „RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers". W: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Red. Dan Jurafsky i in. Online: Association for Computational Linguistics, s. 7567–7578. DOI: 10.18653/v1/2020.acl-main.677. URL: https://aclanthology.org/2020.acl-main.677.

Wang, Xuezhi i in. (2022). „Self-Consistency Improves Chain of Thought Reasoning in Language Models". W: URL: http://arxiv.org/abs/2203.11171.

Xu, Xiaojun, Chang Liu i Dawn Song (2017). „SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning". W: arXiv: 1711.04436 [cs.CL].

Xue, Linting i in. (2021). „mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer". W: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Red. Kristina Toutanova

i in. Online: Association for Computational Linguistics, s. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41.

Yang, Shuo i in. (2023). „Rethinking Benchmark and Contamination for Language Models with Rephrased Samples". W: arXiv: 2311.04850 [cs.CL].

Yu, Tao, Zifan Li i in. (2018). „TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation". W: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Red. Marilyn Walker, Heng Ji i Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, s. 588–594. DOI: 10.18653/v1/N18-2093. URL: https://aclanthology.org/N18-2093.

Yu, Tao, Rui Zhang, Heyang Er i in. (2019). „CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases". W: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Red. Kentaro Inui i in. Hong Kong, China: Association for Computational Linguistics, s. 1962–1979. DOI: 10.18653/v1/D19-1204. URL: https://aclanthology.org/D19-1204.

Yu, Tao, Rui Zhang, Kai Yang i in. (2018a). „Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task". W: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics.

— (2018b). *taoyds/spider: scripts and baselines for Spider*. URL: https://github.com/taoyds/spider (term. wiz. 16. 01. 2024).

Yu, Tao, Rui Zhang, Michihiro Yasunaga i in. (2019). „SParC: Cross-Domain Semantic Parsing in Context". W: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics.

Yu, Yong i in. (2019). „A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures". W: *Neural Computation* 31.7, s. 1235–1270. DOI: 10.1162/neco_a_01199.

Yulianto, Ahmad i Rina Supriatnaningsih (2021). „Google Translate vs. DeepL: A quantitative evaluation of close-language pair translation". W: *AJELP: Asian Journal of English Language and Pedagogy* 9 (2), s. 109–127. DOI: 10.37134/ajelp.vol9.2.9.2021.

Zhong, Victor, Caiming Xiong i Richard Socher (2017a). „Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. W: arXiv: 1709.00103 [cs.CL].

— (2017b). „Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. W: arXiv: 1709.00103 [cs.CL].

A Zmodyfikowane prompty do C3

```
UŻYTKOWNIK:
Given the database schema and polish question, perform the following
actions:
1 - Rank all the tables based on the possibility of being used in the SQL
according to the question from the most relevant to the least relevant,
Table or its column that matches more with the question words is highly
relevant and must be placed ahead.
2 - Check whether you consider all the tables.
3 - Output a list object in the order of step 2, Your output should
contain all the tables. The format should be like:
[
    "table_1", "table_2", ...
]
```

Listing A.1: Zmodyfikowany prompt z C3 służący do znalezienia istotnych tabel

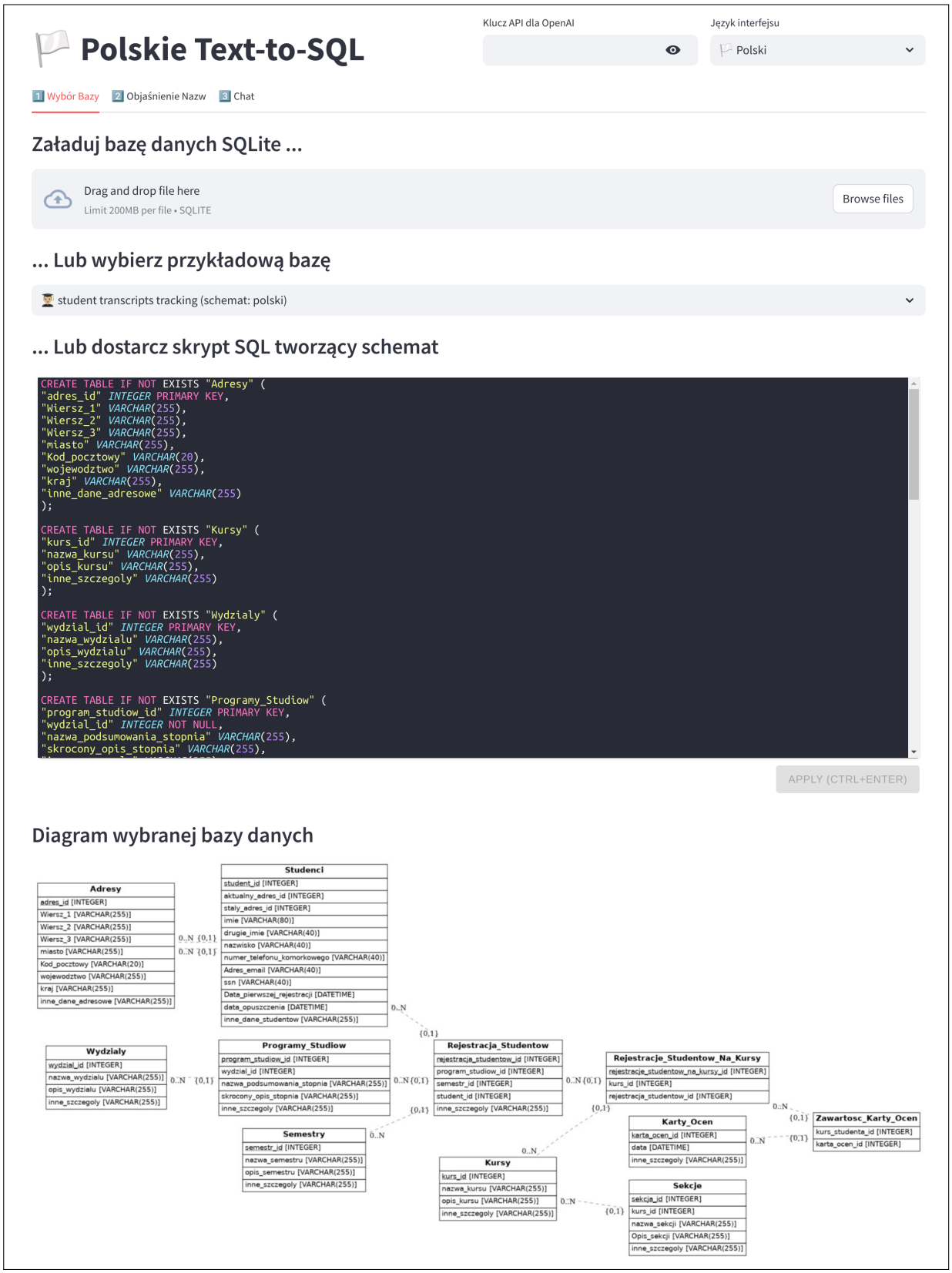
```
UŻYTKOWNIK:
Given the database tables and polish question, perform the following
actions:
1 - Rank the columns in each table based on the possibility of being used
in the SQL, Column that matches more with the question words or the
foreign key is highly relevant and must be placed ahead. You should output
them in the order of the most relevant to the least relevant.
Explain why you choose each column.
2 - Output a JSON object that contains all the columns in each table
according to your explanation. The format should be like:
{
    "table_1": ["column_1", "column_2", ......],
    "table_2": ["column_1", "column_2", ......],
    "table_3": ["column_1", "column_2", ......],
     ......
}
```

Listing A.2: Zmodyfikowany prompt z C3 służący do znalezienia istotnych kolumn

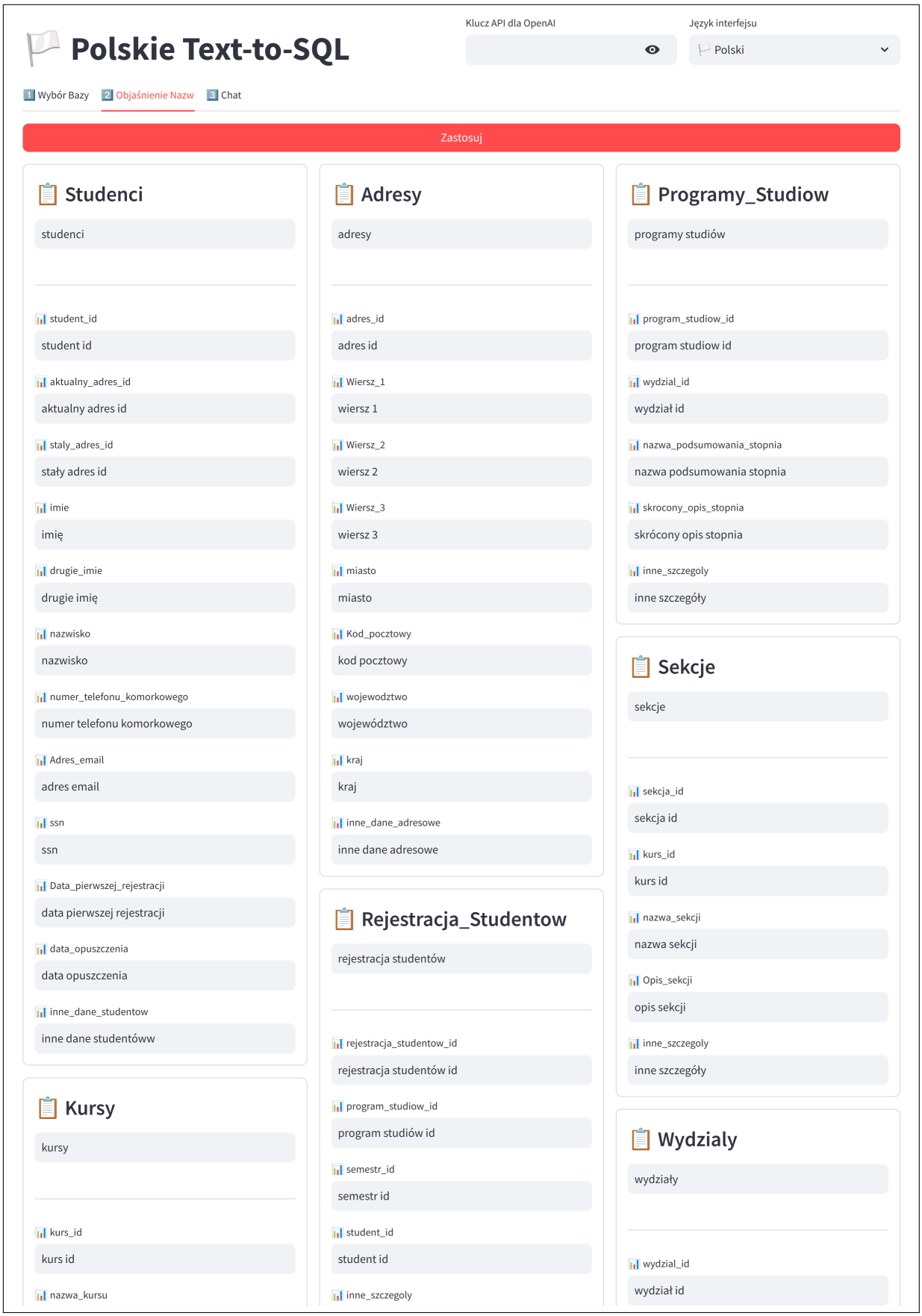
```
SYSTEM:
You are now an excellent SQL writer for polish databases, first I'll give
you some tips and examples, and I need you to remember the tips, and do
not make same mistakes.
UŻYTKOWNIK:
Tips 1:
Question: Które A ma największą liczbę B?
Gold SQL: select A from B group by A order by count ( * ) desc limit 1;
Notice that the Gold SQL doesn't select COUNT(*) because the question only
wants to know the A and the number should be only used in ORDER BY clause,
there are many questions asks in this way, and I need you to remember this
in the the following questions.
ASYSTENT:
Thank you for the tip! I'll keep in mind that when the question only asks
for a certain field, I should not include the COUNT(*) in the SELECT
statement, but instead use it in the ORDER BY clause to sort the results
based on the count of that field.
UŻYTKOWNIK:
Tips 2:
Don't use "IN", "OR", "LEFT JOIN" as it might cause extra results, use
"INTERSECT" or "EXCEPT" instead, and remember to use "DISTINCT" or "LIMIT"
when necessary.
For example,
Question: Jakie A zostały nominowane zarówno do nagrody B jak i do nagrody C?
Gold SQL should be: select A from X where award = 'B' intersect select A
from X where award = 'C';
ASYSTENT:
Thank you for the tip! I'll remember to use "INTERSECT" or "EXCEPT"
instead of "IN", "OR", or "LEFT JOIN" when I want to find records that
match or don't match across two tables. Additionally, I'll make sure to
use "DISTINCT" or "LIMIT" when necessary to avoid repetitive results or
limit the number of results returned.
UŻYTKOWNIK:
### Complete sqlite SQL query only and with no explanation, and do not
select extra columns that are not explicitly requested in the query.
Query will be provided in Polish language.
```

Listing A.3: Zmodyfikowany prompt z C3 służący do generowania zapytania SQL

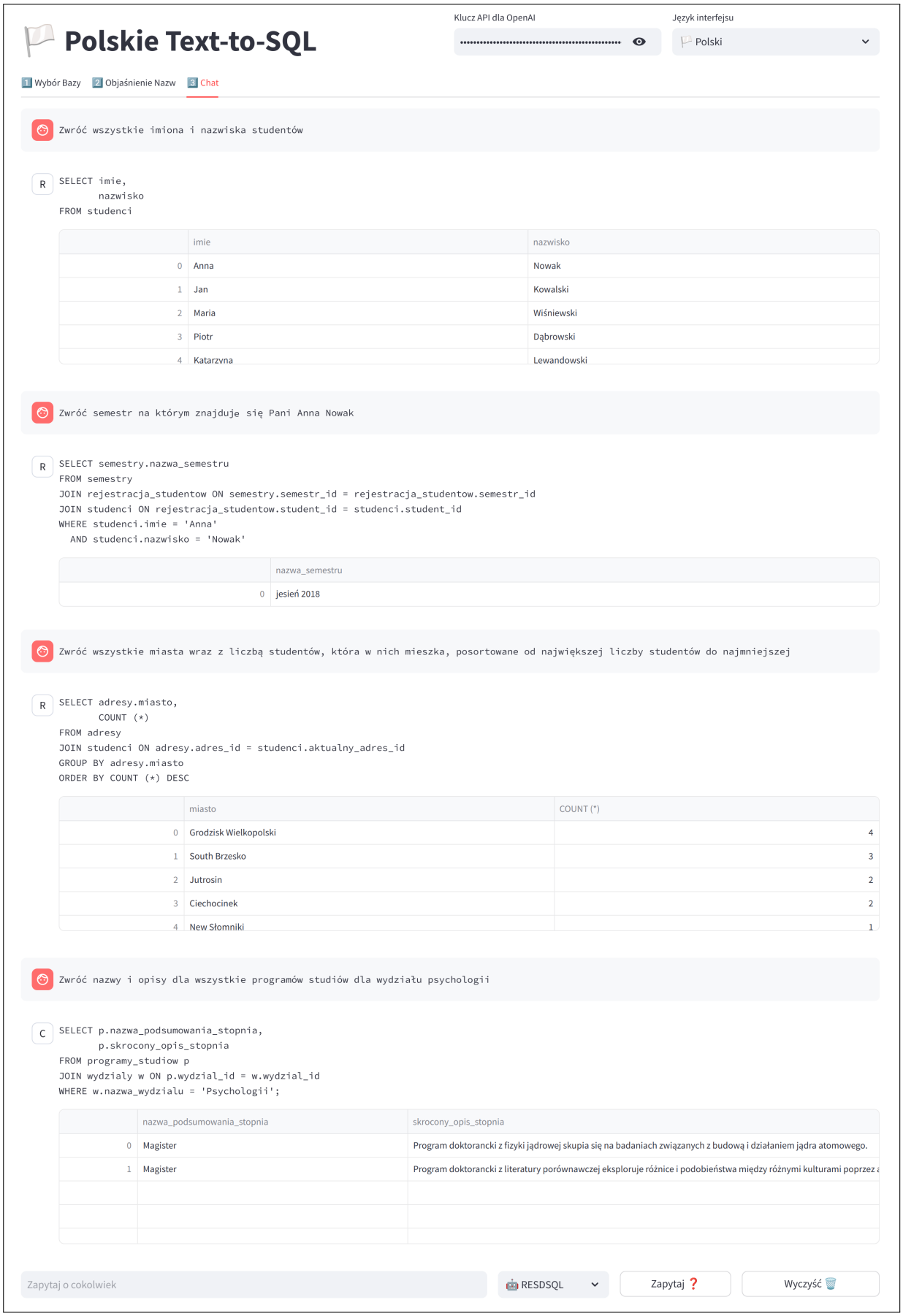
B Interfejs graficzny



Rys. B.1: Zakładka wyboru bazy danych w stworzonym interfejsie graficznym



Rys. B.2: Zakładka objaśnienia nazw schematu w stworzonym interfejsie graficznym



Rys. B.3: Zakładka chatu w stworzonym interfejsie graficznym

Wykaz rysunków

Wykaz tabel

Wykaz listingów

Wykaz skrótów

- **API** – Application Programming Interface
- **ASCII** – American Standard Code for Information Interchange
- **AST** – Abstract Syntax Tree
- **AUC** – Area Under the ROC Curve
- **BERT** – Bidirectional Encoder Representations from Transformers
- **CPU** – Central Processing Unit
- **DK** – Domain Knowledge
- **EM** – Exact Set Match
- **ER** – Entity Relationship
- **EX** – Execution Accuracy
- **GloVe** – Global Vectors for Word Representation
- **GPG** – GNU Privacy Guard
- **GPU** – Graphics Processing Unit
- **HF** – Hugging Face
- **JSON** – JavaScript Object Notation
- **LSTM** – Long Short-Term Memory
- **mT5** – multilingual Text-to-Text Transfer Transformer
- **NER** – Named Entity Recognition
- **NL** – Natural Language
- **NLIDBs** – Natural Language Interfaces for Databases
- **NMT** – Neural Machine Translation
- **RAM** – Random Access Memory
- **RAT** – Relation Aware Transformer
- **SParC** – Semantic Parsing in Context
- **SQL** – Structured Query Language
- **T5** – Text-to-Text Transfer Transformer
- **VRAM** – Video Random Access Memory