1. Вступ
   1. Визначення проблеми t2sql
   2. Лінгвістична складність
   3. Складність розуміння баз даних
   4. Sql синтаксис
   5. Історія розвитку text-to-sql
   6. Існуючі набори даних
   7. Проблеми української мови
2. Теоретична частина

2.1 Опис роботи і фреймворку mac-sql   
2.2 Обгрунтування вибору великої мовної моделі та спосіб доступу  
2.3 Обгрунтування вибору PostgreSQL  
2.4 Обгрунтування способу тестування роботи фреймворку  
2.5 спосіб тестування фреймворку на українській мові

1. Практична частина

3.1 Постановка задачі  
3.2 Розробка MAC-SQL з адаптацією під together api та postgresql  
3.2.1 Розробка копії(чи прототипу) mac-sql  
3.2.2 Реалізація виклику API together.ai   
3.2.3 Реалізація створення схеми бази даних з якою буде працювати фреймворк

3.3 Розробка українського датасету «BIRD-UKR»  
3.3.1 Генерація штучних даних   
3.3.2 Створення пар питання – sql-запит  
3.3.3 Написання скрипта для тестування

4. Результати

4.1 Результати base model  
4.2 Результати фреймворку на BIRD&Spider   
4.3 Висновки про здатність фреймворку та моделі працювати з українською мовою та базами даних

5. Висновок

Додатки

1. **Вступ**

Text-to-SQL, також відома як Natural Language to SQL (NL2SQL), є ключовою технологією у сфері управління даними та їх доступності. В основі цієї технології лежить здатність перекладати запити, виражені природною людською мовою, в команди мови структурованих запитів (SQL), які можуть бути виконані реляційною базою даних. Цей процес перекладу дозволяє користувачам взаємодіяти з базами даних, використовуючи свою повсякденну мову, фактично усуваючи необхідність у глибоких знаннях синтаксису SQL. Першочерговою метою систем Text-to-SQL є *демократизація доступу* до багатства інформації, що міститься в реляційних базах даних, тим самим подолавши розрив між людським спілкуванням і структурованою мовою запитів до баз даних. Ця галузь також визнана в академічному та промисловому контекстах під назвою Natural Language Interface to Databases (NLIDB).

У більш широкому контексті обробки природної мови (NLP) і взаємодії з базами даних, Text-to-SQL займає життєво важливе місце. По суті, це завдання належить до області NLP і вважається специфічною формою семантичного аналізу. Семантичний аналіз включає в себе процес перетворення вхідних даних на природній мові в структуровану, зрозумілу машині логічну форму, причому цільовою логічною формою у випадку Text-to-SQL є SQL. Ця галузь зазнала значної еволюції, значною мірою завдяки значним досягненням у галузі штучного інтелекту, особливо у сферах генеративного штучного інтелекту та великих мовних моделей (LLM). Ці досягнення призвели до помітного підвищення точності та зручності систем Text-to-SQL з плином часу.

Основною проблемою, яку ефективно вирішує Text-to-SQL, є бар'єр, пов'язаний з необхідністю знання SQL для осіб, які не володіють технічними знаннями щодо запитів до баз даних. Пропонуючи інтерфейс на природній мові, вона досягає значної демократизації доступу до даних.

* 1. Визначення Text2SQL(NL2SQL)

Процес перекладу запитів на природній мові в команди SQL зазвичай включає послідовність чітко визначених кроків. Він починається з введення даних користувачем, коли людина формулює запит на своїй природній мові. Після цього до вхідних даних застосовуються методи обробки природної мови (NLP), щоб визначити наміри користувача і контекст запиту. Потім оброблені дані проходять етап розуміння запиту, якому часто сприяють передові моделі машинного навчання, зокрема великі мовні моделі (LLM), які аналізують оброблені компоненти, щоб інтерпретувати основний запит користувача. Кульмінацією цих етапів є генерація SQL, де система створює синтаксично правильний SQL-запит, призначений для точного відображення даних, які користувач хоче отримати.

Ефективність систем Text-to-SQL ґрунтується на кількох ключових компонентах і технологічних основах. Обробка природної мови (Natural Language Processing, NLP) слугує початковим важливим етапом, на якому вона використовується для аналізу природної мови, введеної користувачем, визначення релевантних ключових слів і, зрештою, визначення конкретного наміру, що стоїть за запитом. Великі мовні моделі (LLM), особливо ті, що базуються на архітектурах трансформаторів, таких як BERT і GPT, відіграють ключову роль у розумінні контексту і нюансів, притаманних природній мові, а також у створенні відповідних SQL-запитів. Процес семантичного аналізу є центральним для Text-to-SQL, що включає в себе перетворення запиту на природній мові в машинно-інтерпретоване представлення, яке потім зіставляється з відповідним SQL-запитом. Відображення схеми - це ще один важливий крок, на якому модель встановлює зв'язки між словами і фразами, що використовуються в запиті, і конкретними таблицями, стовпцями бази даних та їх взаємозв'язками. Нарешті, часто застосовуються методи оптимізації запитів, щоб переконатися, що згенерований SQL-запит є не тільки коректним, але й ефективним у виконанні.

По суті, завдання Text-to-SQL - це завдання семантичного розбору, основною метою якого є перетворення запиту користувача на природній мові в логічну форму, яку може зрозуміти і виконати машина, якою в даному контексті є мова SQL. Цей процес вимагає не лише розуміння буквального значення слів, використаних у запиті природною мовою, але й розуміння намірів користувача і того, як ці наміри відображаються на конкретних елементах і операціях, доступних у схемі бази даних.

* 1. Лінгвістична складність

Незважаючи на значний прогрес у технології Text-to-SQL, кілька складних викликів і постійних проблем продовжують перешкоджати її широкому і бездоганному застосуванню. Однією з найголовніших перешкод є неоднозначність, притаманна природній мові. Користувачі часто формулюють свої запити, використовуючи нечіткі терміни, розмовну лексику або вузькоспецифічний жаргон, пов'язаний з конкретною галуззю, який не має прямого або однозначного відповідника в структурованій схемі бази даних. Ця неоднозначність проявляється в різних формах, включаючи лексичну неоднозначність, коли одне слово може мати кілька різних значень залежно від контексту; синтаксичну неоднозначність, коли граматична структура речення допускає більш ніж одну інтерпретацію; і недостатню конкретизацію, коли лінгвістичному виразу не вистачає деталей, щоб чітко передати конкретні наміри користувача. Крім того, складності природної мови поширюються на референційну неоднозначність, коли може бути незрозуміло, до чого відноситься займенник або займенникова фраза, і неоднозначність еліпсису, коли частини речення маються на увазі, але не вказані явно. У контексті запитів до бази даних неоднозначність запиту може виникнути, коли один запит природною мовою може бути інтерпретований кількома семантично різними способами, виходячи зі структури та змісту однієї таблиці бази даних. На додаток до цих лінгвістичних проблем, неоднозначність також може виникати через перекриття імен схем в базі даних або через існування декількох заплутаних шляхів зв'язків між різними об'єктами бази даних.

Значний аспект складності полягає в ефективній обробці лінгвістичних варіацій. Користувачі можуть виражати одну й ту саму інформаційну потребу або мету запиту різними способами, використовуючи різні синоніми, перефразування та граматичні структури. Надійна система Text-to-SQL повинна бути здатна розпізнавати і правильно інтерпретувати ці різноманітні лінгвістичні форми. Крім того, в ідеалі система повинна бути стійкою до недосконалостей у введенні даних користувачем, таких як типові орфографічні та граматичні помилки, які можуть значно ускладнити завдання розуміння природної мови.

* 1. Складність схем баз даних

Ще один значний рівень труднощів виникає через складність і різноманітність схем баз даних, що зустрічаються в реальних додатках. Зазвичай виробничі бази даних містять сотні, якщо не тисячі, таблиць, кожна з яких має численні стовпці, а зв'язки між цими таблицями можуть бути надзвичайно складними. Цей величезний обсяг і складність ускладнюють для систем Text-to-SQL ефективне включення всієї релевантної інформації про схему в єдиний контекст обробки, особливо з огляду на обмеження розмірів вхідних даних моделі. Проблема ще більше ускладнюється тим, що бази даних у різних галузях часто дотримуються дуже різних угод про імена, формати даних і загальні структури таблиць. Це включає використання неінтуїтивно зрозумілих назв стовпців, переважання абревіатур, які можуть бути незрозумілими для всіх, і загальну неоднозначність імен, що вимагає від моделі Text-to-SQL потужних міркувань для правильного виведення зв'язків між таблицями і стовпцями. Крім того, реальні бази даних не є статичними; їхні схеми можуть бути динамічними і розвиватися, що вимагає від систем Text-to-SQL механізмів безперервної адаптації до цих змін.

Іншим важливим аспектом складності є розуміння семантики бази даних. Це виходить за рамки простого знання назв таблиць і стовпців; модель повинна володіти глибоким розумінням всієї схеми бази даних, включаючи типи даних стовпців, зв'язки, визначені первинними і зовнішніми ключами, і загальну логічну структуру, яка з'єднує різні таблиці. Більше того, в ідеалі система повинна мати певний рівень розуміння значення даних, що містяться в цих таблицях, які часто можуть бути вузькоспецифічними і вимагати контекстних знань, що виходять за рамки самої схеми бази даних.

Критично важливим аспектом перетворення Text-to-SQL є зв'язування та вирівнювання схем, що передбачає точне відображення сутностей, згаданих у природномовному запиті, у відповідні таблиці, стовпці та SQL-операції в схемі бази даних. Це основне завдання, яке лежить в основі всього процесу перекладу. Найпоширенішою проблемою є невідповідність між описами природною мовою та реальними назвами стовпців у схемі SQL, яку часто називають «проблемою невідповідності». Навіть з розвитком моделей великих мов, ці моделі все ще не можуть точно узгодити запит користувача з правильними елементами схеми, що іноді призводить до вибору надмірної кількості стовпців або таблиць у спробі охопити всі аспекти вхідних даних.

* 1. SQL syntax

Складність SQL-запитів, які користувачі можуть створювати на основі вхідних даних природною мовою, також становить значну проблему. Хоча прості запити з базовими критеріями відбору часто обробляються з достатньою точністю, створення більш складних SQL-запитів, які включають об'єднання декількох таблиць, вкладені підзапити і складну умовну фільтрацію, залишається складним завданням для багатьох систем. Для багатотабличних запитів система повинна бути здатна точно визначати зв'язки між таблицями, наприклад, за допомогою обмежень зовнішнього ключа, щоб створити відповідні речення JOIN. Крім того, певні домени або специфічні аналітичні вимоги можуть вимагати використання спеціалізованих функцій або операцій SQL, що додає ще один рівень складності до процесу генерації SQL. Навіть такі, здавалося б, прості аспекти SQL, як порядок предикатів у реченні WHERE, що не впливає на результат запиту, можуть стати проблемою для систем Text-to-SQL, які намагаються зіставити природну мову з певним синтаксисом SQL.

Основним завданням перекладу з текстового на SQL є зіставлення природної мови з конкретними конструкціями SQL. Це передбачає правильне визначення ключових слів і речень SQL (таких як SELECT, FROM, WHERE, GROUP BY), які маються на увазі в запиті природною мовою. Крім того, природна мова часто містить порівняльні та вищі ступені порівняння (наприклад, «менше», «найвищий»), які необхідно перевести у відповідні оператори порівняння SQL (<, >, =) та агрегатні функції (SUM, AVG, MAX, MIN, COUNT). Складність значно зростає при роботі з вкладеними запитами (запит всередині іншого запиту) та операціями над множинами (такими як UNION, INTERSECT, EXCEPT), які вимагають більш глибокого розуміння логічної структури запиту.

* 1. Проблема навчання

Здатність систем Text-to-SQL добре працювати в різних сценаріях, відома як узагальнення, а також їхня ефективність при застосуванні до баз даних або доменів, для яких вони не були спеціально підготовлені (так звані міждоменні завдання), залишаються серйозними проблемами. Моделі, які досягають високої точності на конкретних навчальних наборах даних, часто не можуть підтримувати цю продуктивність, коли стикаються з новими, небаченими базами даних або варіаціями того, як користувачі можуть формулювати свої запити. Варіації мовних виразів, такі як використання абревіатур або синонімів, які не зустрічалися під час навчання, можуть призвести до помітного зниження продуктивності системи. Завдання стає особливо складним, коли ми маємо справу з міждоменними сценаріями, які вимагають від системи роботи з різноманітними темами та структурами баз даних, прикладом чого є еталонні набори даних на кшталт Spider.

Якість і доступність даних, що використовуються для навчання моделей Text-to-SQL, також відіграють вирішальну роль у їхній загальній ефективності. Відсутність великих, високоякісних навчальних наборів даних, особливо для конкретних доменів або складних типів запитів, може суттєво знизити продуктивність цих систем. Крім того, існуючі еталонні набори даних можуть мати внутрішні обмеження з точки зору різноманітності SQL-запитів, які вони містять, діапазону операторів SQL, які вони використовують, і того, наскільки реалістично вони відображають складність реальних баз даних. Навіть у межах однієї бази даних невідповідності у форматуванні даних можуть створювати додаткові проблеми для систем Text-to-SQL, які намагаються інтерпретувати запити користувачів.

* 1. Історія розвитку

Розвиток технології Text-to-SQL охоплює кілька десятиліть, позначених значними змінами в підходах і можливостях, що відображають більш широкі досягнення в галузі штучного інтелекту та обробки природної мови. Перші спроби подолати розрив між людською мовою і взаємодією з базами даних почалися з ранніх систем, заснованих на правилах, в епоху до глибокого навчання. Ці системи, що з'явилися в 1960-х і 1970-х роках, включаючи такі відомі приклади, як BASEBALL, LUNAR, LIFER/LADDER і CHAT-80, покладалися на ретельно розроблені граматичні правила та евристики для перекладу запитів на природній мові у виконувані команди SQL. Хоча ці системи продемонстрували початкову перспективність, особливо в контексті простих і вузькоспецифічних баз даних, вони зіткнулися зі значними обмеженнями, коли зіткнулися з більш складними запитами або базами даних, що характеризуються різноманітними схемами, які постійно розвиваються. Це було значною мірою пов'язано з притаманною негнучкістю правил, визначених вручну, і значними зусиллями, необхідними для функціональної інженерії, щоб адаптувати ці системи до нових доменів. Такі системи, як LUNAR і NaLIX, демонструючи потенціал семантичного синтаксичного аналізу, страждали від недостатньої масштабованості та адаптивності до ширших застосувань.

У міру того, як бази даних зростали як у розмірі, так і в складності, галузь стала свідком переходу до статистичних методів і методів машинного навчання. Ці підходи використовували алгоритми, які могли вивчати основні відповідності між вхідними даними на природній мові та відповідними їм SQL-запитами з даних. Порівняно зі своїми попередниками, що базувалися на правилах, ці системи, засновані на машинному навчанні, демонстрували здатність обробляти ширший спектр більш складних запитів. Критично важливим компонентом цих методів був інжиніринг ознак, який передбачав ручне вилучення релевантних ознак як з вхідного тексту природною мовою, так і зі схеми бази даних для навчання алгоритмів навчання.

Основним поворотним моментом в історії Text-to-SQL стала поява глибокого навчання. Впровадження рекурентних нейронних мереж (RNN), зокрема мереж з довгою короткочасною пам'яттю (LSTM), ознаменувало значний стрибок вперед у можливостях цієї галузі. Моделі Sequence-to-sequence (Seq2Seq), що часто використовують архітектуру кодера-декодера, побудовану на основі LSTM, стали популярною основою для безпосереднього перекладу запитів природною мовою в SQL-послідовності. Для подальшого підвищення продуктивності цих моделей були інтегровані механізми уваги, що дозволяють декодеру зосереджуватися на найбільш релевантних частинах вхідної послідовності при генерації кожної частини вихідного SQL-запиту.

Потім ця галузь перейшла в еру Трансформер-моделей та Попередньо навчених Мовних Моделей (ПНММ). Архітектури на основі Трансформерів, такі як BERT, GPT та T5, з того часу стали домінуючим підходом у дослідженнях та застосунках Text-to-SQL. Їхній успіх значною мірою зумовлений винятковою здатністю розуміти контекстуальні зв'язки як у запиті природною мовою, так і в схемі бази даних за допомогою механізмів самоуваги. Ключовою технікою цього періоду стало попереднє навчання цих великих мовних моделей на величезних обсягах загальних текстових даних, з подальшим їх доналаштуванням (fine-tuning) на специфічних наборах даних Text-to-SQL. Ця парадигма значно підвищила продуктивність систем Text-to-SQL. Такі моделі, як RAT-SQL та TaBERT, були спеціально розроблені для ефективної інтеграції інформації про схему бази даних у процес навчання моделі, що призвело до підвищення точності.

Графові Нейронні Мережі (ГНМ) також все частіше застосовуються в Text-to-SQL для ефективного кодування структури та зв'язків у схемі бази даних. Представляючи схему у вигляді графа, ГНМ можуть формувати насичені представлення, що покращують розуміння моделлю контексту бази даних. RAT-SQL – це помітна модель, яка використовує трансформерну мережу, що враховує зв'язки (relation-aware), для кодування схеми та зв'язування, демонструючи ефективність цього підходу.

Останнім часом у цій галузі спостерігається швидка поява та впровадження Великих Мовних Моделей (LLM), таких як GPT-4 та Gemini, які здійснили подальшу революцію у можливостях систем Text-to-SQL. Ці моделі, завдяки своїй величезній здатності до розуміння та генерації людиноподібного тексту, продемонстрували вміння обробляти все складніші SQL-запити, сформовані на основі природномовних запитів. Поточні дослідження в цій галузі часто досліджують використання навчання в контексті (in-context learning), де ВММ надають кілька прикладів для скеровування генерації SQL без необхідності явного доналаштування ваг моделі, а також традиційні підходи до доналаштування, за яких ВММ навчаються на специфічних наборах даних Text-to-SQL для адаптації до завдання.

Історичний розвиток Text-to-SQL демонструє чітку тенденцію переходу від систем, що базувалися на чітко визначених правилах, які вимагали значних ручних зусиль і мали обмежену сферу застосування, до все більш досконалих моделей глибокого навчання, здатних вивчати складні закономірності безпосередньо з даних. Останні досягнення, зумовлені Трансформер-моделями та, особливо, великими мовними моделями, ознаменували значний зсув парадигми в цій галузі. Ці сучасні підходи дозволили значно покращити точність та здатність систем Text-to-SQL ефективно обробляти складніші та міждоменні сценарії, наближаючи втілення ідеї справді природномовної взаємодії з базами даних.

* 1. Набори даних

На розробку та оцінювання систем Text-to-SQL значно вплинула наявність еталонних наборів даних (бенчмарків) та метрик, що використовуються для оцінки продуктивності. Ранні дослідження часто спиралися на однодоменні набори даних, такі як ATIS (для інформації про авіаперельоти) та GeoQuery (для географічних запитів), які забезпечували сфокусований контекст для оцінки початкових підходів.

Значним поворотним моментом у цій галузі стала поява міждоменних бенчмарків, метою яких було оцінити здатність моделей до узагальнення на різних схемах баз даних та типах запитів. WikiSQL був одним із перших великомасштабних наборів даних такого типу, що містив відносно прості SQL-запити до великої кількості таблиць, взятих з Вікіпедії. Після WikiSQL було представлено Spider — складніший та більш вимогливий міждоменний набір даних, що містить різноманітний набір SQL-запитів, включно із запитами зі з'єднаннями (joins) та вкладеними структурами, для колекції баз даних з різних доменів. Для завдань, що включають діалогову взаємодію з базами даних, були створені набори даних, такі як SParC та CoSQL, для оцінки здатності моделей обробляти багатоходові діалоги.

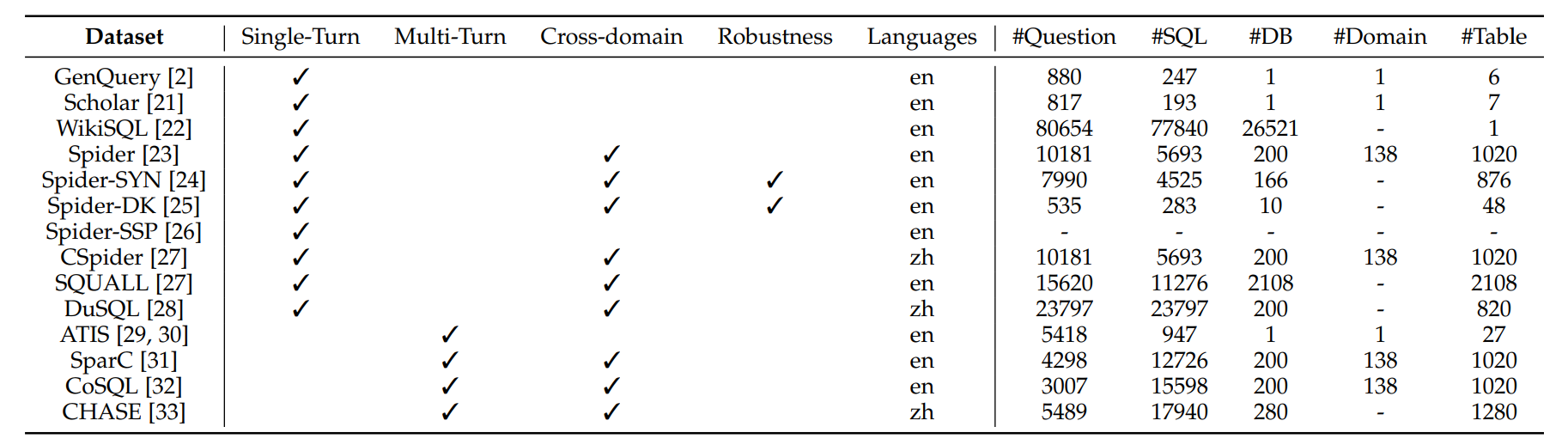
На додаток до цих базових наборів даних, були розроблені різноманітні спеціалізовані та доповнені набори даних для вирішення специфічних проблем або оцінки окремих аспектів систем Text-to-SQL. Прикладами є KaggleDBQA, SEDE та EHRSQL, які мають на меті представити більш реалістичні сценарії, а також набори даних, такі як Spider-DK, Spider-Syn та Spider-Realistic, які вносять варіації в оригінальний набір даних Spider для перевірки стійкості моделі до знань предметної області, використання синонімів та відсутності явних назв стовпців у запитаннях. Крім того, докладалися зусилля для створення доповнених наборів даних, таких як GRAPPA, GAP2SQL та Spider-CG, які генерують синтетичні навчальні приклади для збільшення розміру та різноманітності навчальних даних і потенційного покращення продуктивності моделей Text-to-SQL.

Продуктивність систем Text-to-SQL зазвичай оцінюється за допомогою кількох ключових метрик. Точність виконання (Execution Accuracy, EX) – це критично важлива метрика, яка вимірює, чи згенерований SQL-запит при виконанні в базі даних повертає правильну відповідь. Точна відповідність (Exact Match, EM) – ще одна поширена метрика, яка перевіряє, чи згенерований SQL-запит точно збігається (посимвольно) з еталонним (анотованим людиною) SQL-запитом. Точність логічної форми (Logical Form Accuracy, LF) має на меті оцінити семантичну еквівалентність згенерованого SQL, часто шляхом порівняння їхніх абстрактних синтаксичних дерев, що може бути більш поблажливим до синтаксичних варіацій, які не змінюють значення запиту.

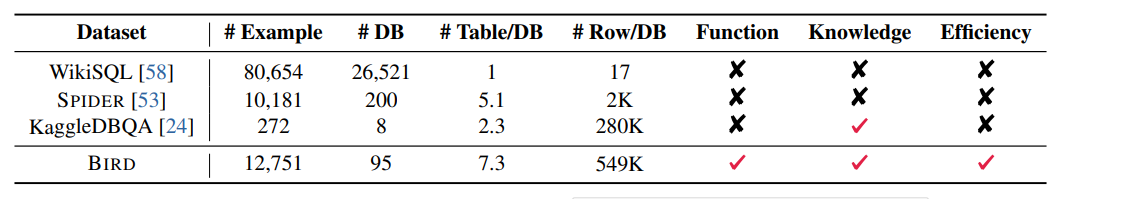
Незважаючи на прогрес, досягнутий завдяки цим наборам даних та метрикам, зростає усвідомлення їхніх обмежень. Існуючі бенчмарки можуть не повністю охоплювати різноманітність SQL-запитів, що зустрічаються в реальних застосунках, повний спектр операторів SQL, складність реальних схем баз даних або нюанси природномовних запитань, поставлених користувачами. Метрики оцінки, такі як точна відповідність, можуть бути надто суворими, штрафуючи згенеровані запити, які семантично еквівалентні еталонним, але відрізняються синтаксично. Точність виконання, хоч і важлива, іноді може призводити до хибнопозитивних спрацювань, коли неправильний SQL-запит може випадково повернути правильний результат для певного стану бази даних. Отже, існує постійна потреба в розробці бенчмарків та метрик оцінки, які б точніше відображали виклики реальних завдань Text-to-SQL та справжні наміри користувацьких запитів.

Бенчмарк BIRD (BIg Bench for LaRge-Scale Database Grounded in Text-to-SQLs) був створений як великий еталон для оцінки Text-to-SQL у реальних застосунках. Згідно з описом, BIRD містить 12 751 складний приклад запитів до інформації з 95 великих баз даних загальним розміром 33,4 ГБ, що охоплюють 37 професійних доменів. Для навчання та розробки його творці зібрали та модифікували 80 реляційних баз даних з відкритим кодом з реальних аналітичних платформ (Kaggle, Relation.vit). Щоб додатково уникнути витоку даних, було підготовлено 15 додаткових реляційних баз даних для прихованого тестового набору. На основі цих баз даних для збору питань природною мовою та відповідних SQL-запитів використовувався краудсорсинг. Додатково, автори бенчмарку запропонували нову метрику оцінки — Valid Efficiency Score (VES) — для оцінки ефективності згенерованих SQL-запитів. За їхніми твердженнями, BIRD є першим бенчмарком Text-to-SQL, який враховує ефективність, сприяючи використанню більш ефективних методів запитів у контексті великих обсягів даних та "зашумлених" значень у базах даних.

Розвиток наборів даних у галузі Text-to-SQL відіграв ключову роль у стимулюванні прогресу можливостей моделей. Перехід від однодоменних до все складніших міждоменних бенчмарків спонукав дослідників розробляти моделі, здатні обробляти більш реалістичні та складні сценарії. Однак поточна дискусія щодо обмежень існуючих бенчмарків та метрик оцінки підкреслює необхідність постійного вдосконалення способів оцінки продуктивності та реальної застосовності систем Text-to-SQL.



[1 <https://arxiv.org/pdf/2208.13629>]



[2 https://arxiv.org/pdf/2305.03111]

* 1. Стан на сьогодні

Сильні сторони цих сучасних передових моделей, особливо тих, що базуються на LLM(ВММ?), включають їхню надійну здатність до перенесення знань (transferability), що дозволяє їм легше адаптуватися до нових завдань та доменів. Вони також демонструють покращені здібності до логічного мислення, що дає їм змогу обробляти складніші SQL-завдання, які виходять за межі обмежень попередньо визначених фреймворків. Техніки, такі як промптинг "Ланцюжок Думок" (Chain-of-Thought, COT), де модель явно генерує проміжні кроки міркувань, та декомпозиція завдань, де складні запити розбиваються на менші, легші для керування частини, сприяли покращенню логіки та точності згенерованого SQL.

Незважаючи на ці помітні досягнення, сучасні моделі Text-to-SQL все ще стикаються з низкою властивих їм обмежень та викликів, коли йдеться про реальні застосунки. Однією з постійних проблем є обізнаність щодо схеми (schema awareness). Великі мовні моделі потребують глибокого розуміння відповідної схеми бази даних, щоб генерувати точні SQL-запити.

Навіть із передовими моделями згенерований SQL іноді може бути неточним через такі проблеми, як галюцинації ШІ (генерація неправильного або безглуздого SQL), використання помилкових назв стовпців, неправильне розуміння схеми бази даних або погано сформульовані промпти у випадку ВММ.

Крім того, здатність сучасних моделей до узагальнення на небачених раніше даних, особливо нових баз даних та доменів, які не були включені в їхнє навчання, все ще є важливою сферою для вдосконалення.(це проблема будь-яких моделей машинного навчання?)

Додатково, інтеграція систем Text-to-SQL у реальні середовища викликає занепокоєння щодо вразливостей безпеки, таких як потенційна можливість атак SQL-ін'єкцій, якщо система не розроблена ретельно з урахуванням безпеки.

Подолання цих обмежень, особливо в таких сферах, як розуміння складних схем баз даних, ефективна обробка властивої природній мові неоднозначності, покращення узагальнення на нові та небачені дані, а також забезпечення безпеки та ефективності цих систем, буде вирішальним для ширшого впровадження та впливу технології Text-to-SQL.

2.1 MAC-SQL[3 https://arxiv.org/abs/2312.11242] вибір

Фреймворк MAC-SQL був розроблений для подолання цих обмежень шляхом застосування мультиагентного підходу (Xu et al., стор. 1, 3). Він складається з трьох спеціалізованих агентів, які взаємодіють для генерації та уточнення SQL запитів (Xu et al., стор. 2, Рис. 2):

1. **Агент Selector:** Призначений для роботи з великими схемами баз даних. Він аналізує запит користувача та повну схему для ідентифікації та відбору лише релевантних таблиць і стовпців. Це дозволяє зменшити вхідний контекст для LLM, знижуючи ймовірність помилок, пов'язаних із зайвою інформацією (Xu et al., стор. 3, розділ 3.2).
2. **Агент Decomposer:** Використовує підхід "ланцюжка думок" (chain-of-thought) для декомпозиції складного запиту користувача на послідовність простіших підзапитів. Для кожного підзапиту генерується відповідний SQL. Цей підхід полегшує моделі обробку запитів, що вимагають складних логічних міркувань (Xu et al., стор. 3, розділ 3.3; стор. 4, Рис. 3).
3. **Агент Refiner:** Застосовує зовнішні інструменти для перевірки синтаксичної коректності та виконуваності згенерованого SQL запиту. У разі виявлення помилки або отримання неочікуваного результату (наприклад, порожній набір даних), Refiner використовує отриманий зворотний зв'язок для корекції SQL запиту (Xu et al., стор. 4, розділ 3.4; стор. 4, Рис. 4).

**Емпірична Ефективність**

Ефективність архітектури MAC-SQL підтверджується результатами експериментів, представленими в роботі Xu et al. Комбінація MAC-SQL з моделлю GPT-4 досягла точності виконання (Execution Accuracy) 59.59% на тестовому наборі бенчмарку BIRD, що на момент публікації дослідження було передовим результатом (state-of-the-art) (Xu et al., стор. 1, 6). Це свідчить про переваги запропонованого мультиагентного підходу для вирішення складних завдань Text-to-SQL.

Важливим фактором при виборі MAC-SQL є його доступність для дослідницької спільноти та розробників.

Надання доступу до коду фреймворку через репозиторій GitHub (Xu et al., стор. 1, виноска 1) значно спрощує процес відтворення результатів, адаптації та інтеграції фреймворку в інші проекти. Це усуває залежність від пропрієтарних моделей та сприяє подальшому розвитку напрямку Text-to-SQL.

2.2 Base model(?)

Вибір Базової Мовної Моделі: Llama 3.1 70B

В оригінальному дослідженні фреймворку MAC-SQL (Xu et al.) як основна велика мовна модель (LLM), що забезпечувала функціонування агентів Selector, Decomposer та Refiner, використовувалася модель GPT-4 від OpenAI. У даній роботі, як базова LLM для всіх агентів фреймворку MAC-SQL, була обрана модель **Llama 3.1 70B**. Цей вибір обумовлений кількома ключовими перевагами порівняно з використанням GPT-4:

1. Відкритий Доступ та Гнучкість**:** Llama 3.1 70B є моделлю з відкритим доступом (з відповідною ліцензією), на відміну від GPT-4, яка доступна лише через пропрієтарний API. Це надає значно більший контроль над моделлю та процесом її використання:
   * Глибоке донавчання (Fine-tuning): Можливість донавчати модель безпосередньо на специфічних даних та завданнях агентів MAC-SQL (спрощення схеми, декомпозиція, генерація/корекція SQL) дозволяє точніше адаптувати її поведінку до конкретних вимог задачі Text-to-SQL, потенційно покращуючи точність та релевантність результатів.
   * Прозорість та Відтворюваність: Використання конкретної версії моделі з відкритими вагами підвищує прозорість дослідницького процесу та спрощує відтворення результатів іншими дослідниками, оскільки немає залежності від можливих змін у закритій API-моделі.
   * Гнучкість Розгортання: Відкриті моделі можуть бути розгорнуті локально або у приватній хмарі, що забезпечує повний контроль над інфраструктурою та конфіденційністю даних, що є важливим при роботі з чутливими базами даних.
2. Економічна Ефективність: Використання GPT-4 через API пов'язане зі значними операційними витратами, особливо при обробці великих наборів даних або при ітеративному процесі розробки та тестування, що також було відзначено як обмеження у оригінальному дослідженні (Xu et al., стор. 7). Локальне розгортання та використання Llama 3.1 70B усуває ці витрати, пов'язані з API-викликами, роблячи процес розробки та експлуатації значно економічно ефективнішим.
3. Висока Продуктивність: Llama 3.1 70B належить до останнього покоління потужних відкритих LLM, демонструючи високу продуктивність у широкому спектрі завдань, включаючи розуміння природної мови, генерацію коду та логічні міркування, які є критичними для Text-to-SQL. Хоча GPT-4 показала передові результати в оригінальній роботі MAC-SQL, сучасні відкриті моделі, такі як Llama 3.1 70B, мають потенціал досягти порівнянної або навіть вищої продуктивності, особливо після цільового донавчання, при цьому зберігаючи переваги відкритості та контролю.

2.3 Вибір Цільової Бази Даних для Тестування: PostgreSQL

Вибір середовища для виконання та тестування згенерованих SQL запитів є важливим аспектом оцінки систем Text-to-SQL. Хоча SQLite часто використовується в академічних бенчмарках (наприклад, для перевірки виконання в деяких частинах Spider або BIRD) завдяки своїй простоті інтеграції та відсутності необхідності в окремому сервері, у даній роботі для валідації та оцінки точності виконання (Execution Accuracy) було обрано реляційну систему керування базами даних (РСКБД) PostgreSQL.

Цей вибір обумовлений наступними міркуваннями щодо обмежень SQLite та переваг PostgreSQL для цілей ретельного тестування:

1. Обмеженість Функціональності SQLite:
   * SQLite, будучи вбудованою, файловою базою даних, має спрощений діалект SQL порівняно з повнофункціональними серверними РСКБД, такими як PostgreSQL, MySQL, Oracle або SQL Server, які зазвичай використовуються в корпоративних (enterprise) середовищах.
   * SQLite може не підтримувати або мати іншу реалізацію розширених функцій SQL, таких як віконні функції, Common Table Expressions (CTE), складні типи даних, розширені оператори JOIN, специфічні функції для роботи з датами/часом, текстом чи JSON, а також складні обмеження (constraints). Це означає, що SQL запит, який коректно виконується в SQLite, може бути невалідним або повертати інші результати в більш потужній РСКБД.
   * SQLite часто є менш суворим щодо типізації даних та синтаксису запитів порівняно з PostgreSQL.
2. Переваги PostgreSQL для Реалістичного Тестування:
   * Репрезентативність для підприємницьких Сценаріїв: PostgreSQL є однією з найпоширеніших РСКБД з відкритим кодом, що широко використовується в промислових та корпоративних системах. Тестування на PostgreSQL забезпечує більш реалістичну оцінку здатності системи генерувати SQL, придатний для використання в типових робочих середовищах.
   * Багатший та Строгіший Діалект SQL: PostgreSQL підтримує широкий спектр стандартних та розширених функцій SQL. Вимога генерації коректного SQL для PostgreSQL ставить перед системою Text-to-SQL складніше завдання, змушуючи її враховувати більшу кількість синтаксичних та семантичних нюансів, що веде до більш надійної оцінки її можливостей.
   * Ретельна Валідація: Успішне виконання складного запиту в PostgreSQL, з його строгим дотриманням стандартів та підтримкою розширених можливостей, надає вищий рівень впевненості у коректності та загальній придатності згенерованого SQL порівняно з тестуванням на більш обмеженій платформі як SQLite.

2.4 Вибір Тестових Наборів Даних: BIRD (minidev) та Spider

Для всебічної оцінки ефективності фреймворку MAC-SQL на основі Llama 3.1 70B та перевірки його здатності генерувати коректні SQL запити для PostgreSQL, було обрано два ключових бенчмарки: Spider та піднабір BIRD (minidev). Вибір цих наборів даних дозволяє оцінити різні аспекти продуктивності системи Text-to-SQL.

1. Spider(***не дуже велике бажання включати***):
   * Призначення: Spider (Yu et al., 2018) є широко визнаним стандартним бенчмарком для оцінки можливостей систем Text-to-SQL щодо узагальнення на різних доменах (cross-domain generalization). Він містить велику кількість баз даних (200) з різних предметних областей, але самі бази даних, як правило, є відносно невеликими за розміром та обсягом даних.
   * Основні Виклики: Головний акцент Spider робиться на складності структури запитів та необхідності коректно інтерпретувати схему бази даних (назви таблиць, стовпців, зв'язки між ними) для відповіді на запитання користувача. Він перевіряє здатність моделі генерувати складні SQL конструкції (вкладені запити, агрегації, JOINи по багатьох таблицях) незалежно від специфіки домену.
   * Обґрунтування Вибору: Включення Spider до процесу тестування є необхідним для порівняння продуктивності розробленого рішення з попередніми роботами та для оцінки його фундаментальної здатності розуміти різноманітні структури баз даних та перекладати запити на SQL без глибокого аналізу *змісту* даних. Це дозволяє оцінити ефективність, зокрема, агента Decomposer у роботі зі складними запитами.
2. BIRD (minidev Subset):
   * Призначення: Бенчмарк BIRD (Li et al., 2023) був створений для подолання обмежень Spider та моделювання реалістичних сценаріїв взаємодії з великими базами даних. Він фокусується на викликах, пов'язаних із значними обсягами даних, "брудними" даними (dirty values), потребою в зовнішніх знаннях та ефективністю SQL запитів.
   * Основні Виклики: BIRD вимагає від систем Text-to-SQL не лише розуміння схеми, але й здатності міркувати над вмістом бази даних (database content reasoning), обробляти великі схеми (десятки таблиць) та генерувати запити, які є не тільки коректними, але й потенційно ефективними при виконанні на великих обсягах даних.
   * Використання minidev: Для даного дослідження було використано піднабір minidev з BIRD, що містить 500 запитів. Вибір цього піднабору є компромісом між потребою оцінити систему на реалістичних даних та практичними обмеженнями щодо часу та ресурсів, необхідних для повного тестування на BIRD. Піднабір minidev, хоча й менший за повний набір розробки (dev set), все ще містить репрезентативні приклади викликів, характерних для BIRD, включаючи великі схеми та запити, що вимагають аналізу вмісту.
   * Обґрунтування Вибору: Включення BIRD (minidev) є критично важливим для оцінки здатності фреймворку MAC-SQL справлятися саме з тими проблемами, на вирішення яких він був спрямований — робота з великими схемами (тестування агента Selector), складне багатоетапне міркування (Decomposer) та потенційна потреба в корекції запитів через складну взаємодію з даними (Refiner). Це дозволяє оцінити продуктивність системи в сценаріях, максимально наближених до реальних умов експлуатації.

2.5 Створення та Використання Українського Тестового Набору Даних

Мотивація та Необхідність

Стандартні бенчмарки для Text-to-SQL, такі як Spider та BIRD, розроблені переважно для англійської мови. Це створює прогалину в оцінці ефективності систем Text-to-SQL для інших мов, зокрема для української. Оскільки однією з цілей даної роботи є розробка та оцінка системи, здатної обробляти запити українською мовою, виникла необхідність у створенні відповідного тестового набору даних. Пряме застосування англомовних бенчмарків не дозволяє оцінити здатність моделі коректно інтерпретувати лінгвістичні особливості, термінологію та структуру запитів українською мовою в контексті взаємодії з базами даних.

Процес Створення "Silver Dataset"

Для перевірки роботи фреймворку MAC-SQL на українській мові було створено спеціальний тестовий набір даних, який ми називаємо “BIRD-UKR”. Цей датасет був розроблений шляхом штучної генерації даних та пар питання--SQL-запит. Для генерації даних було переважно використано Claude 3.7 Sonnet(Thinking)[4 <https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf>]. Для генерації пар запит-SQL було обрано Gemini 2.5 PRO[5 <https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#building-on-best-gemini>], що ***(те що геміні дуже гарно генерує sql бо гугл мабуть fine-tune’ли їх для свого google cloud) (no source)***

Процес включав наступні кроки:

1. Аналіз BIRD: Було вивчено структуру та характеристики бенчмарку BIRD, зокрема:
   * Типи запитів та їх складність (необхідність JOIN-ів, агрегацій, вкладених запитів, аналізу вмісту даних).
   * Складність схем баз даних (кількість таблиць, зв'язків).
   * Реалістичність та різноманітність сценаріїв.
2. Розробка Сценаріїв та Запитів: На основі аналізу BIRD було розроблено нові сценарії та відповідні їм природномовні запити безпосередньо українською мовою. Ці запити були сформульовані так, щоб відображати аналогічний рівень складності та типи викликів, що присутні в BIRD (наприклад, запити, що вимагають міркування над значеннями в БД, обробку потенційно великих схем тощо).
3. Створення/Адаптація Схем та Даних: Для підтримки розроблених українських запитів могли бути створені нові схеми баз даних або адаптовані існуючі, потенційно з використанням даних, релевантних для українського контексту.

Важливо зазначити, що отриманий датасет класифікується як "срібний" (silver), а не "золотий" (gold). Це означає, що він є корисним для оцінки, але може містити певні артефакти перекладу або не повністю відображати все різноманіття того, як носії української мови формулювали б запити до баз даних органічно.

Мета Використання в Тестуванні

Включення українського срібного датасету до процесу тестування дозволяє:

1. Оцінити Розуміння Української Мови: Перевірити здатність базової моделі (Llama 3.1 70B) коректно інтерпретувати семантику та синтаксис українських запитів у контексті наданої схеми бази даних.
2. Валідувати Роботу Фреймворку: Переконатися, що логіка агентів Selector, Decomposer та Refiner в рамках MAC-SQL залишається ефективною при зміні мови вхідного запиту.
3. Виявити Специфічні Виклики: Ідентифікувати потенційні труднощі, пов'язані з обробкою української мови (наприклад, багатозначність слів, відмінювання, специфічна термінологія).
4. Забезпечити Базову Оцінку: Надати кількісну оцінку продуктивності розробленої системи безпосередньо для українськомовних сценаріїв.

3 ПРАКТИЧНА ЧАСТИНА

Я манав