Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка застосунку для генерації SQL-запитів, заданих природною мовою
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-5

Артем Кальченко
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект

(повна назва освітньої програми)
Керівник доц. Олександра Вітько
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кальченку Артему Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для генерації SQL-запитів, заданих природною мовою

затверджена наказом університету від 19 травня 20 25 р. № 378Ст
2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 20 25 р.
3. Вихідні дані до роботи Документація бібліотеки transformers, документація фреймворку Flask, документація фреймворку Next.js, документація Python, наукові статті на сайті Medium, набір даних «synthetic_text_to_text»

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Постановка задачі
3) Теоретичні дослідження
4) Практичні дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	19.05.2025 – 20.05.2025	виконано
3	Постановка задачі	20.05.2025 – 21.05.2025	виконано
4	Теоретичні дослідження	21.05.2025 – 04.06.2025	виконано
5	Практичні дослідження	04.06.2025 – 10.06.2025	виконано
6	Розробка застосунку	10.06.2025 – 15.06.2025	виконано
7	Оформлення пояснювальної записки	15.06.2025 – 17.06.2025	виконано
8	Захист перед ЕК	17.06.2025	

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Олександра Вітько
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 69 с., 27 рис., 3 дод., 34 джерела.

БД, ГЕНЕРАЦІЯ ТЕКСТУ, ГЕНЕРАЦІЯ SQL, НЕЙРОННІ МЕРЕЖІ, ОБРОБКА ПРИРОДНОЇ МОВИ, BART, BERT, GPT, NLP, SQL, T5.

Об’єкт дослідження – процес генерації SQL-запитів за допомогою генеративних нейронних мереж на основі запитів, сформульованих природною мовою.

Предмет дослідження – програмний застосунок, що автоматизує побудову SQL-запитів із текстових вказівок користувача.

Мета роботи – дослідити методи донавчання претренованих трансформерних моделей для генерації SQL-коду, провести їх порівняльний аналіз за відповідними метриками, а також реалізувати вебзастосунок, що інтегрує ці моделі для спрощення взаємодії з реляційними БД.

Методи дослідження включають: аналіз наукової літератури та сучасних рішень у сфері генеративного ШІ, експериментальне донавчання моделей, оцінювання продуктивності за метриками ROUGE, BLEU, METEOR, тестування функціональності застосунку.

У ході роботи було проаналізовано та підготовлено датасет, виявлено його дисбаланс та проаналізовано наслідки цього. Проведено порівняння сучасних моделей, визначено переваги та недоліки архітектур encoder-only, decoder-only та encoder-decoder. На основі аналізу обрано модель T5 як найбільш придатну для генерації SQL-коду.

Здійснено донавчання обраної моделі, оптимізацію її гіперпараметрів та оцінювання результатів. У якості практичного результату розроблено браузерний клієнт-серверний застосунок, що реалізує інтерфейс для взаємодії користувача з нейронною моделлю генерації SQL-запитів.

ABSTRACT

Bachelor’s thesis contains: 69 pp., 27 fig., 3 ann., 34 references.

BART, BERT, DATABASES, GPT, NATURAL LANGUAGE PROCESSING, NEURAL NETWORKS, NLP, SQL, SQL GENERATION, T5, TEXT GENERATION

Research Object – the process of generating SQL queries using generative neural networks based on natural language queries.

Research Subject – a software application that automates SQL query construction from user instructions expressed in natural language.

Objective – to investigate fine-tuning methods for pretrained transformer models for SQL code generation, perform a comparative analysis of trained models using relevant metrics, and develop a web application that integrates these models to facilitate user interaction with relational databases.

Research Methods include analysis of scientific literature and recent advances in generative AI, experimental fine-tuning of models, performance evaluation using ROUGE, BLEU, and METEOR metrics, functional testing of the developed application.

During the study, the dataset was analyzed and prepared, revealing imbalance and analyzed the consequences of this. A comparative evaluation of contemporary models was conducted, highlighting advantages and disadvantages of encoder-only, decoder-only, and encoder-decoder architectures. Based on this analysis, the T5 model was selected as the most suitable for SQL generation tasks.

Fine-tuning of the selected model was performed, including hyperparameter optimization and result evaluation. As a practical outcome, a browser-based client-server application was developed to provide a user interface for interaction with the neural network model for SQL query generation.

ЗМІСТ

Вступ 7
1 Аналіз предметної галузі 8
1.1 Актуальність 8
1.2 Специфіка мови SQL 8
1.3 Аналіз обраного набору даних 10
2 Постановка задачі 17
3 Теоретичні дослідження 19
3.1 Загальні положення 19
3.2 Обґрунтування вибору моделі 21
3.2.1 Будова encoder-only моделей 21
3.2.2 Будова decoder-only моделей 30
3.2.3 Будова encoder-decoder моделей 33
3.2.4 Висновки з вибору моделі 36
3.3 Вибір метрик оцінювання 36
3.3.1 BLEU 38
3.3.2 ROUGE 39
3.3.3 METEOR 41
3.3.4 Висновки з вибору метрик 43
4 Практичні дослідження та розробка застосунку 44
4.1 Запропоновані методи покращення навчання моделі 44
4.2 Навчання моделей 47
4.3 Дослідження результатів оцінки T5 52
4.4 Створення інтерфейсу застосунку 56
Висновки 60
Перелік джерел посилання 62
Додаток А Діаграма розподілу тематик в тренувальному наборі 67
Додаток Б Приклади входжень колонки sql_task_type 68
Додаток В Відомість кваліфікаційної роботи 69

ВСТУП

У сучасному цифровому середовищі доступ до інформації, що зберігається у базах даних, є критично важливим для підтримки ефективного прийняття рішень. Проте, формування SQL-запитів буває рутинною задачею, яка виснажує спеціалістів, використовуючи їх розумові здібності, які можуть бути використані для більш кваліфікованих завдань.

У зв’язку з цим зростає актуальність інструментів, що можуть спростити доступ до даних за допомогою природної мови. Генеративні моделі, побудовані на основі трансформерної архітектури, демонструють високий потенціал у задачах перетворення природної мови в SQL-запити. Однак, такі моделі розглядаються не як заміна досвідчених фахівців з баз даних, а як інтелектуальні помічники, здатні автоматизувати рутинні дії, прискорити роботу та знизити поріг входу для нових користувачів.

У цій роботі досліджується процес донавчання генеративної моделі, яка виконує трансформацію запитів природною мовою у SQL-запити. Основна увага приділяється аналізу існуючих моделей, їх адаптації до прикладної задачі та оцінці якості генерації в контексті коректності синтаксису та семантики сформованих запитів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Актуальність

Зростання обсягів даних та широке використання реляційних баз даних у різних сферах діяльності вимагають ефективних засобів для швидкого та точного доступу до інформації. Традиційний спосіб взаємодії з базами даних передбачає використання SQL, що потребує знання синтаксису, структури даних та специфічних команд. Це створює бар’єри для користувачів, які не мають достатнього рівня технічної підготовки. Впровадження генеративної нейронної мережі, яка автоматично формує SQL-запити на основі природномовних запитів, дозволить значно розширити коло осіб, які можуть ефективно працювати з базами даних, забезпечуючи зручний і доступний інтерфейс.

1.2 Специфіка мови SQL

Темою роботи є генерація SQL-запитів, заданих природною мовою. SQL (Structured query language) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, а також для організації системи контролю за доступом до бази даних.

Більшість дій, які потрібно виконати в базі даних, виконуються за допомогою SQL-інструкцій. Як і інші мови програмування, SQL має унікальний синтаксис для сприймання цих самих інструкцій машиною. Так як і в інших мовах програмування в SQL є певні ключові слова, які слугують набором команд. Ключові слова є регістронезалежними, проте прийнято писати їх у верхньому регістрі для покращення читання.

Ключове слово «SELECT» слугує для вибору полів із бази даних. Таким чином, інструкція «SELECT field1» вказує на те, що треба обрати колонку із назвою «field1».

Також, доволі уживаним є ключове слово «DISTINCT», яке використовується після ключового слова «SELECT» та слугує для уникнення результатів що повторюються. Оператор «FROM» використовується для визначення конкретної таблиці, з якої будуть узяті колонки, вказані оператором «SELECT».

Ключове слово «WHERE» слугує для визначення умови, під яку підходять певні дані таблиці. Також для більш точної фільтрації можна використовувати оператори роботи над множинами, такими як «AND» та «OR».

Крім цього, важливим оператором є «ORDER BY» для визначення порядку подання елементів запиту, відносно значень колонки або колонок. Ключове слово «DESC» відобразить список в спадаючому порядку, «ASC» – навпаки.

У синтаксисі SQL також є чимало агрегаційних функцій. Наприклад, «SUM», яка повертає суму масиву, або «AVG», яка обчислює середнє значення. Також, присутні такі функції як «MIN» та «MAX», які призначені для пошуку мінімуму та максимуму, відповідно. Для пошуку кількості екземплярів використовується «COUNT». Ці функції можна використовувати для повернення агрегованого результату, наприклад, завдяки запиту «SELECT COUNT (*) FROM table» можна отримати кількість комірок таблиці.

Крім цього, доволі важливим для представлень ключовим словом є «LIMIT». Воно використовується, якщо потрібно повернути лише перші декілька входжень.

Іноді зручніше дати певному результату запиту локальне ім’я. Для цього використовується «AS». Через нього можна визначити псевдоніми для таблиць, стовпців, отриманих констант тощо.

Крім цього, важливим оператором є «GROUP BY». Він використовується для групування певних рядків за спільним значенням. Разом з ним часто використовують агрегатори щоб отримати агреговане значення для певної групи. Важливим зауваженням є те, що із групами не працює ключове слово «WHERE», для фільтрації груп використовується «HAVING»

Для групування таблиць між собою використовується «JOIN» із вказанням типу об’єднання. Найлегше буде продемонструвати принцип на діаграмах Венна, вони наведені на рисунку 1.1.

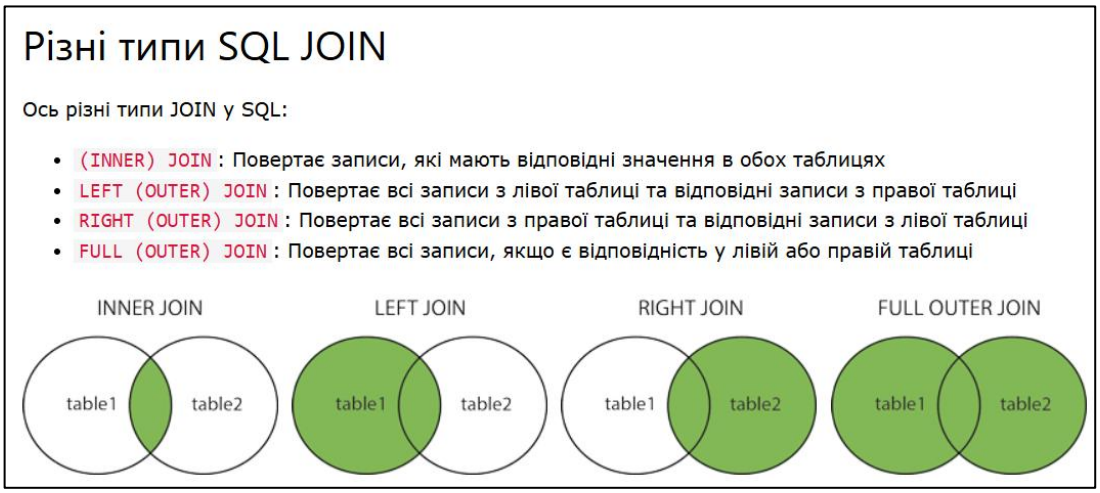

Рисунок 1.1 – Діаграми Венна, що демонструють принципи об’єднання таблиць [1]

1.3 Аналіз обраного набору даних

Логічним є припущення, що схеми баз даних не завжди однакові, тому вони є необхідною метаінформацією для моделі, яка подаватиметься на вхід. Для цієї задачі було обрано датасет «synthetic_text_to_text» на платформі Hugging Face.

Навіть з назви можна зробити висновок, що цей датасет є синтетичним, тобто створеним штучно. Файл, доданий до опису цього

датасету, це підтверджує: датасет було створено за допомогою Gretel Navigator – нейронної мережі для генерації синтетичних даних на основі текстових запитів. В даних міститься 105,851 входжень, які поділені на тренувальний і тестовий. Тренувальний містить 100,000 входжень, тестовий – 5,851.

Датасет має 11 полів. id – унікальний ідентифікатор. domain – тематика, до якої належить запит (aerospace, finance, healthcare тощо). domain_description – опис того, що включає ця тематична область Наприклад: дані про пацієнтів, інформація про польоти. sql_complexity – складність запиту. Хоча цей параметр і має відображати складність, він радше вказує на тип запиту. Серед його значень – single join, aggregation, basic SQL, window functions, subqueries, multiple_joins, set operations та CTEs. sql_complexity_description – обґрунтування інформації, зазначеної в параметрі sql_complexity. sql_task_type – тип завдання, яке виконується. Серед значень цього параметра – analytics and reporting, data manipulation тощо. sql_task_type_description – обґрунтування зазначеного типу завдання. sql_prompt – запит природною мовою (англійською). sql_context – контекст запиту, тобто схема бази даних для цього запиту. sql – SQL-запит, сформований на основі тексту природною мовою та контексту. sql_explanation – пояснення до SQL-запиту.

Усі поля, крім поля id, мають текстовий тип (string), id – цілочисельне (integer).

Поле domain має 100 унікальних значень. Відповідно, domain_description також має 100 унікальних описів – по одному для кожного значення. sql_complexity має лише 8 унікальних значень, і sql_complexity_description – також. sql_task_type містить 4 унікальні значення типів завдань, а sql_task_type_description – 4 відповідні описи.

Загальну інформацію про датасет наведено на рисунку 1.2.

	count	unique
domain	100000	100
domain_description	100000	100
sql_complexity	100000	8
sql_complexity_description	100000	8
sql_task_type	100000	4
sql_task_type_description	100000	4
sql_prompt	100000	100000
sql_context	100000	89766
sql	100000	99271
sql_explanation	100000	99777

Рисунок 1.2 – Загальна інформація щодо колонок тренувального датасету

Було проведено аналіз розподілу кількості входжень у кожну підгрупу, утворену унікальними значеннями характеристик sql_task_type, sql_complexity та domain. Усі інші колонки або мають виключно унікальні значення, або їхній розподіл прямо корелює з цими категоріями – з огляду на зазначені вище причини. Відповідні діаграми наведено на рисунках 1.3–1.5.

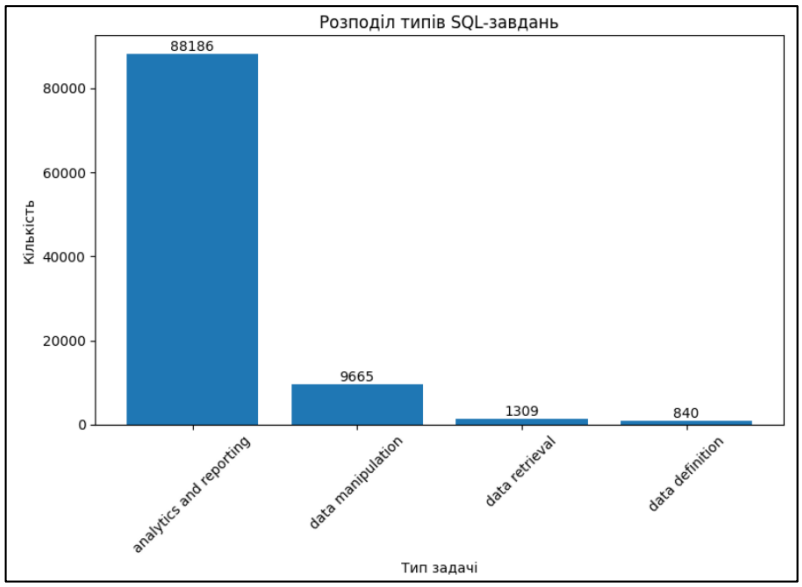

Рисунок 1.3 – Розподіл колонки sql_task_type

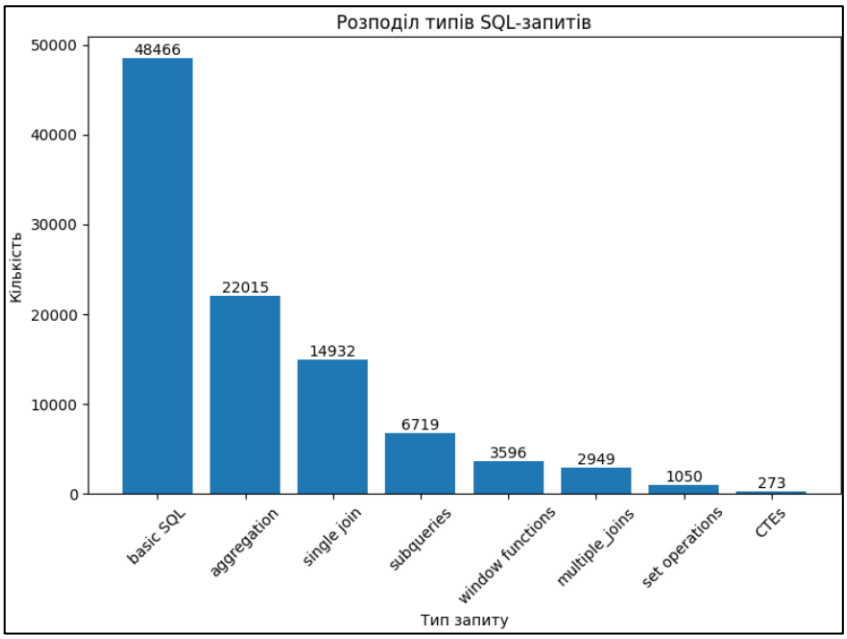

Рисунок 1.4 – Розподіл колонки sql_complexity

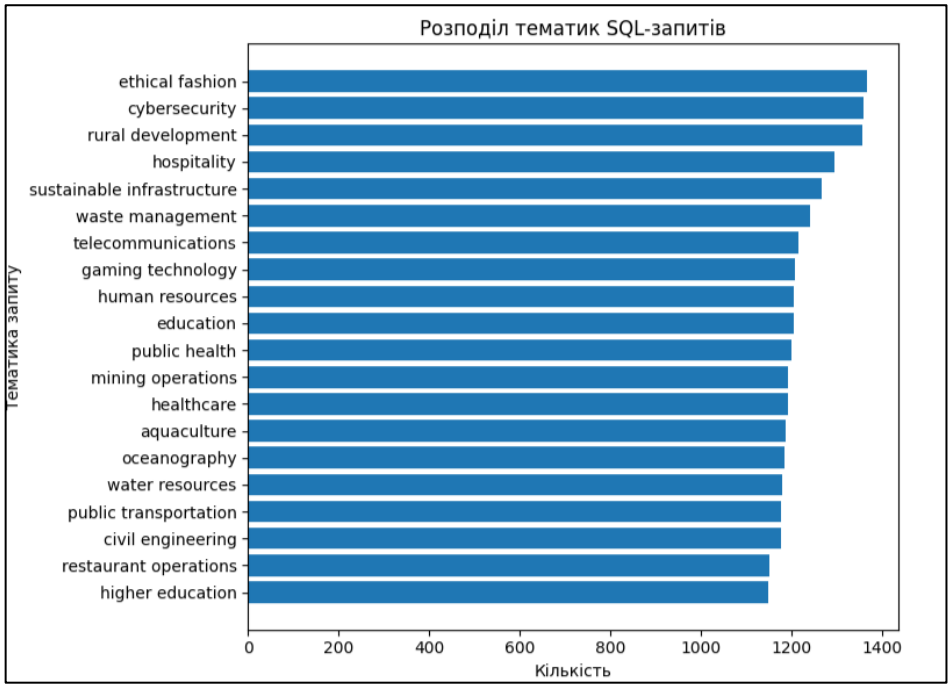

Рисунок 1.5 – Діаграма розподілу перших 20-ти значень параметру domain

Колонка domain має 100 унікальних значень, тому відображати їх усі на одному рисунку було б недоречно. У зв’язку з цим було прийнято рішення показати 20 найбільших за чисельністю значень – їх наведено на

рисунку 1.5. Водночас у додатку А представлено діаграму розподілу для кожної з тематик.

Як можна побачити на рисунку 1.3, найбільш частими є входження із типом завдань «analytics and reporting». При чому, входжень саме цього типу більше, приблизно, в 7.5 разів ніж інших в сумі. Тобто, очевидним буде сказати, що датасет незбалансований. Розберемо детальніше що це означає для моделі. Колонка sql_task_type відображає тип задачі, для якої було створено запит. Приклади кожного з типів завдання наведено на рисунку 1.6. Більша кількість прикладів наведено в додатку Б.

А
```sql
SELECT salesperson_id, name, SUM(volume) as total_volume
FROM timber_sales
JOIN salesperson
ON timber_sales.salesperson_id = salesperson.salesperson_id
GROUP BY salesperson_id, name
ORDER BY total_volume DESC;
```

Б
```sql
UPDATE carbon_offsets
SET end_date = '2025-12-31'
WHERE initiative_name = 'Initiative 1' AND country = 'Australia';
```

В
```sql
CREATE TABLE artifacts (id INT, artifact_type VARCHAR(25
material VARCHAR(255), analysis_date DATE);
```

Г
```sql
SELECT * FROM defense_diplomacy
WHERE participant_country IN ('China', 'India');
```

Рисунок 1.6 – Приклади запитів кожного типу задачі: А – analytics and reporting, Б – data manipulation, В – data definition, Г – data retrieval

Як можна побачити, запити із типом задачі analytics and reporting є запитами для вибору даних із БД. На перший погляд, може здатись, що запити типу data retrieval також відносяться до analytics and reporting і було припущено помилки проте, при детальному огляді можна помітити, що ключовою відмінністю двох типів є відсутність в data retrieval будь-якої обробки, або агрегації даних. Тобто, суто певні рядки таблиці. Запити типу

задачі data manipulation, виходячи з назви, представляють собою зміну значень колонок для певних входжень, додавання окремих входжень, їх видалення. data definition, в свою чергу, представляють собою запити маніпулювання таблицею: видалення таблиці, її створення, зміна типів даних, додавання нових колонок тощо.

Як було зазначено раніше, датасет є незбалансованим, що безпосередньо впливає на якість навчання моделі: вона, ймовірно, генеруватиме запити з маніпуляціями з таблицею або з даними менш точно.

Крім того, модель може ускладнювати запити через наявність великої кількості прикладів із попередньою обробкою даних. Тобто вона, ймовірно, використовуватиме агрегацію або групування з нульовим ефектом – лише для того, щоб отримати просте представлення потрібних рядків.

Водночас імовірним є й те, що моделі буде достатньо прикладів для навчання побудові простих запитів. Це пов’язано з тим, що в подібних запитах зазвичай спостерігається обмежена кількість комбінацій ключових слів, тож модель може ефективно навчитися навіть на наявному обсязі даних.

З рисунка 1.4 також видно, що датасет є незбалансованим і за типами самих запитів. Тип basic SQL має майже стільки входжень, скільки всі інші типи разом узяті. Це також не є позитивним чинником, оскільки модель може навчитися генерувати окремі типи запитів гірше за інші.

Проаналізувавши тематики запитів, розподіл яких наведено на рисунку 1.5 та в додатку А, можна зробити висновок, що датасет є незбалансованим і за цим параметром. Проте в цьому випадку різниця не є суттєвою.

До того ж, це параметр, який відображає виключно тематику запиту – тобто сферу діяльності, до якої його можна віднести. Як відомо, сфера діяльності впливає, у кращому разі, на назви змінних або таблиць, але жодним чином не на синтаксис самого запиту. Відтак можна припустити,

що цей дисбаланс матиме мінімальний вплив на здатність моделі розуміти SQL.

Окремої уваги заслуговує колонка sql_context. Запити в ній подано у вигляді SQL-інструкцій зі створення таблиць, а в деяких випадках – також із додаванням до них даних. Приклад такого запиту наведено на рисунку 1.7.

```
CREATE TABLE creative_ai (application_id INT, name TEXT, region TEXT,
explainability_score FLOAT);
INSERT INTO creative_ai (application_id, name, region,
explainability_score)
VALUES (1, 'ApplicationX', 'Europe', 0.87);
```

Рисунок 1.7 – Приклад входження колонки sql_context

Варто наголосити, що інформація про зв'язки (зовнішні ключі), а також про первинні ключі не зберігається. Відповідно, користувач не вводитиме її через застосунок. Обов'язковими для введення даними є: таблиця (або кілька таблиць), колонки, типи даних цих колонок, іноді входження.

Під час формування запитів до моделі слід враховувати структуру рядка в колонці sql_context. Необхідно уважно дотримуватись синтаксису – ключових слів, пунктуації тощо – інакше модель втратить частину своєї ефективності.

2 ПОСТАНОВКА ЗАДАЧІ

Розроблений застосунок включатиме генеративну нейронну мережу, яка аналізуватиме введені користувачем запити природною мовою та перетворюватиме їх на коректні SQL-інструкції. Коректність інструкцій забезпечуватиметься шляхом перевірки запиту на синтаксичну правильність одразу після його генерації та перед виведенням користувачеві. У разі виявлення синтаксичної помилки модель спробує згенерувати новий запит. Після п’яти невдалих спроб система сповістить користувача про проблему та запропонує перефразувати запит.

Перед початком розробки застосунку необхідно визначитися з моделлю, яка буде донавчатися під конкретне завдання. Крім цього, важливим етапом є аналіз метрик для оцінювання якості моделі, а також порівняння підходів до її навчання та різних архітектур між собою.

Інтерфейс застосунку передбачатиме візуальне представлення згенерованого SQL-запиту з можливістю його редагування перед виконанням. Запит виконуватиметься безпосередньо в підключеній базі даних, а результати відображатимуться в інтерфейсі із можливістю їх експорту у файл. Очевидно, надзвичайно важливою вимогою є забезпечення безпеки даних – саме тому застосунок жодним чином не матиме змоги змінювати оригінальний файл бази даних.

Попри всю потужність і потенціал сучасних нейронних мереж, досягнення 100% точності є наразі неможливим. Саме тому роботу із моделлю організовано поетапно:

1) користувач завантажує схему своєї бази даних;

2) користувач завантажує дані таблиць;

3) користувач вводить запит природною мовою;

4) запит обробляється нейронною мережею, яка генерує відповідний SQL-запит;

5) користувач перевіряє правильність згенерованого запиту;

6) після підтвердження правильності запиту, він виконується.

Таким чином, механізм роботи застосунку передбачає ручну перевірку користувачем правильності згенерованого SQL-коду з огляду на очікуваний результат. Такий підхід має зменшити кількість запитів, які не повертають результат або повертають неправильні дані.

Варто також зазначити, що одним із ключових аспектів у цій роботі є здатність нейронної мережі генерувати запити для будь-якої схеми бази даних. Тому особливу увагу приділено варіативності схем у навчальному датасеті та здатності моделі адекватно інтерпретувати природну мову.

Насамкінець важливо підкреслити, що розроблений застосунок жодним чином не претендує на заміну роботи фахівців будь-якого рівня. Програма позиціонується як допоміжний інструмент для полегшення простої, але рутинної роботи із SQL-запитами.

3 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ

3.1 Загальні положення

Штучні нейронні мережі, які зазвичай просто називають нейронними мережами, або нейромережами – це обчислювальні системи, натхнені біологічними нейронними мережами, які складають мозок тварин.

На базовому рівні нейронна мережа складається з нейронів і зв’язків між ними. Кожен нейрон приймає числові значення з попереднього шару (або з вхідних даних, якщо йдеться про перший шар), обчислює зважену суму цих значень, застосовує активаційну функцію і передає результат далі. Зв’язки між нейронами мають ваги, які визначають вплив одного нейрона на інший у наступному від нього шарі.

Нейронна мережа складається з вхідного шару, одного або кількох прихованих шарів та вихідного шару. Вхідний шар приймає дані, а вихідний формує результат обчислень.

Активаційні функції – це математичні функції, які визначають вихідне значення нейрона на основі зваженої суми його входів. Вони додають до моделі нелінійність, що дозволяє нейромережі моделювати складні залежності.

Активаційні функції поділяють на компресійні та ті, що не обмежують діапазон вихідних значень. Компресійні функції стискають вихід у межах певного інтервалу. Наприклад, sigmoid(x) завжди повертає значення в межах [0, 1], а tanh(x) – у межах [-1, 1], де x – зважена сума входів нейрона.

Компресійні функції часто використовуються у вихідному шарі нейромережі, наприклад, у задачах бінарної класифікації, де sigmoid(x) інтерпретується як ймовірність належності об’єкта до певного класу.

Найпростішим представником нейронної мережі є одношаровий перцептрон. Його зображено на рисунку 3.1.

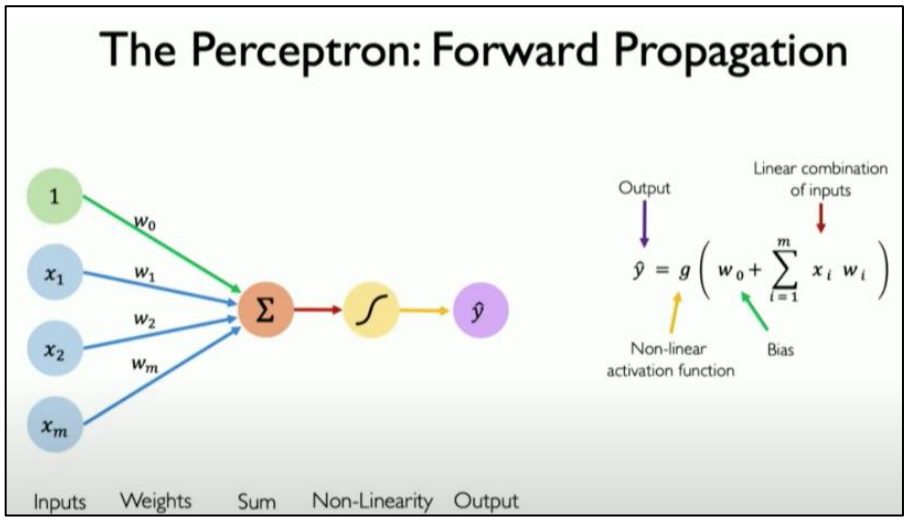

Рисунок 3.1 – Схематичне зображення одношарового перцептрону [5]

Як можна побачити на рисунку 3.1, вхідний шар має деякі значення даних x_m на вхідному шарі, які отримані із вибірки, та константу 1 із неконстантним значенням ваг. Константа також називається біас (bias). Біас слугує для того, щоб нейронна мережа могла краще підлаштувати значення активаційної функції під задачу. Активаційна функція g, яка схематично нагадує графік сигмоїди, не завжди є функцією сигмоїди, проте в більшості випадків є нелінійною.

Завдяки своїй гнучкості нейромережі можна адаптувати до широкого спектра задач – класифікації, регресії, генерації даних тощо – зазвичай із незначними змінами в архітектурі. Проте при переході до нової задачі модель, як правило, потребує перенавчання або донавчання.

Нейронні мережі знайшли широке використання саме у NLP (Natural Language Processing) завданнях, тобто у завданнях обробки природної мови. Серед завдань NLP виділяють: перефразування (paraphrasing), машинний переклад (machine translation), узагальнення (summarization), генерацію тексту тощо.

Також завдання розподіляються за форматами, залежно від входу та виводу, тобто виду інформації, яку отримує модель, і тієї, яка очікується. результатом обчислень моделі. Серед основних форматів завдань в

генеративних моделях із використанням природної мови виділяють: text-to-text, text-to-image, text-to-music, text-to-video тощо. Дана робота присвячена text-to-text формату, а саме text-to-sql – окремому його випадку.

3.2 Обґрунтування вибору моделі

Кожна генеративна нейронна мережа має свої особливості, в тому числі в архітектурі. На загальному рівні архітектури поділяються на: encoder-only, decoder-only, encoder-decoder.

3.2.1 Будова encoder-only моделей

Encoder-only-архітектура дозволяє нейронній мережі «розуміти» текст. Тобто модель має змогу закодувати текст у токени і обробити їх певним чином задля виділення з них придатної для роботи інформації. До завдань таких моделей входять класифікація текстів, виділення інформації тощо. Найвідомішими представниками таких моделей є BERT та RoBERTa.

BERT (Bidirectional Encoder Representations from Transformers) вперше реалізував ідею двостороннього (бідірекційного) кодування контексту, що дозволяє моделі враховувати як попередні, так і наступні слова при обробці тексту, завдяки чому досягається глибше розуміння значення. Окрім цього, при навчанні BERT використовував завдання Next Sentence Prediction, за допомогою якого модель вчиться встановлювати логічні зв’язки між реченнями. RoBERTa, у свою чергу, є покращеною версією BERT, яка відмовилася від Next Sentence Prediction, натомість зосередившись на глибшому маскуванні токенів і навчанні на більших обсягах даних, що дало змогу досягти ще кращих результатів у багатьох NLP-завданнях.

На рисунку 3.2 схематично зображено структуру більшості encoder-only моделей.

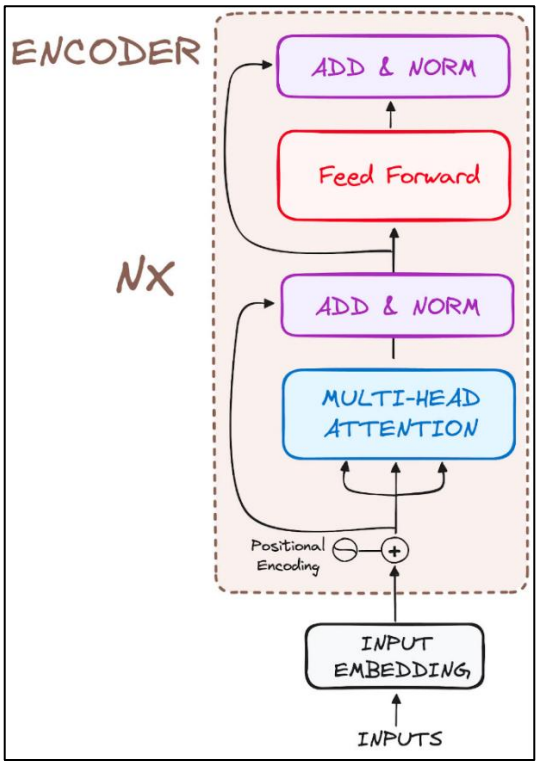

Рисунок 3.2 – Схематичне зображення encoder-only моделей [14]

Як можна побачити, енкодер як окрема структура повторюється N разів. Так, наприклад, BART-base має 12 шарів енкодера, а BART-large – 24.

Розглянемо детальніше будову моделі. На рисунку 3.2 послідовність обробки йде знизу вгору. Першим кроком є створення ембедингів слів (word embeddings). Це пов’язано з тим, що модель не може працювати безпосередньо з текстом у «сирому» вигляді – ці дані потребують попереднього перетворення.

У випадку з природною мовою спершу кожному унікальному слову (токену) присвоюється унікальний індекс на основі заздалегідь сформованого словника. Далі послідовність тексту перетворюється на послідовність відповідних індексів, яка потім конвертується у вектори – ембединг-представлення слів.

Отримана матриця ембедингів має розмірність (T, E), де T – довжина вхідної послідовності (кількість токенів), а E – розмір ембединг-

вектора (embedding dimension). Розмірність E, як правило, задається вручну на етапі побудови моделі.

Словник токенів, який використовується для знаходження ембедингів, також представлений у вигляді матриці розміром (V, E), де V – розмір словника, тобто кількість унікальних токенів. Зазвичай словник формують на основі великих текстових корпусів, щоб якомога більше слів мали відповідне числове представлення, а отже могли бути оброблені без помилок.

Наступним кроком є додавання позиційного кодування (positional encoding). Оскільки трансформери, на відміну від рекурентних нейронних мереж (RNN), обробляють усю послідовність одночасно, а не покроково, вони не мають вбудованої інформації про порядок слів у реченні. Щоб компенсувати це, до вхідних векторів додається позиційне кодування, яке дозволяє моделі враховувати розташування кожного слова у послідовності.

Врахування позиції досягається завдяки створенню вектора, розмірність якого дорівнює розмірності вектора слова в послідовності, після чого отримані вектори додаються для визначення зсуву певного слова. Це дозволяє різним за позицією словам сприйматися по-різному, а однаковим – однаково.

Для знаходження парних індексів позиційного вектора використовується синус, для непарних – косинус. Формула знаходження відповідних позицій наведена на формулах 3.1 та 3.2.

$$\mathrm{PE}_{(\mathrm{pos},2\mathrm{i})} = sin\left(\frac{pos}{10000^{2i/d}}\right), \tag{3.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right), \tag{3.2}$$

де PE – значення позиційного вектора;

pos – значення позиції слова у реченні (0 – перше, 1 – друге тощо);

i – індекс виміру позиційного вектора;

d – розмірність позиційного вектора.

Функції косинусів та синусів дозволяють кодувати багатовимірні простори однаково ефективно для будь-якої кількості вимірів. Число 10000 із ступенем в знаменнику дозволяє «розтягувати» ці функції, завдяки чому значення для кожного з вимірів є унікальними.

Наступним етапом обробки є Multi-Head Attention. Цей механізм дозволяє розглянути послідовність, умовно, декількома «головами». Для його розуміння перш за все слід розібратися, що таке механізм self-attention, оскільки саме його використовує кожна з голів.

Перш за все, слід зазначити, що класичними підходами до створення векторних представлень слів у NLP є кількісна векторизація, TF-IDF-векторизація, One-Hot Encoding, Bag-Of-Words тощо. Головною проблемою цих підходів є незбереження сенсу речення. Модель, навчена за допомогою таких підходів, погано вловлює сенс слів, оскільки векторний простір, у якому знаходяться ці слова, не є самодостатнім. Це проявляється в тому, що вектори, схожі за сенсом, не обов’язково розташовані близько один до одного. Аналогічно, слова з протилежним значенням (антоніми) можуть знаходитися поруч.

Задля вирішення цієї проблеми було запропоновано створювати ембедінги слів. Таким чином, утворені вектори розташовуватимуться ближче один до одного (часто для вимірювання схожості застосовують функцію cosine similarity, яка найкраще передає подібність слів через обчислення косинуса кута між відповідними векторами). Проте навіть такі покращення не дають ідеального результату в передачі сенсу для моделі.

Так, для людини є очевидним, що в словосполученнях «apple pie» та «apple store» слово «apple» має різне значення: у першому випадку – це начинка пирога, у другому – товар магазину. Аналогічно і зі словом «light», яке може перекладатися як «світло» або як «легкий», але без перекладу це

можна визначити лише за контекстом, який модель не розуміє, адже векторне представлення цих слів фактично однакове.

Для розв'язання цієї проблеми було створено механізм self-attention, а результатом його роботи стали contextual embeddings. Цей механізм утворений трьома фундаментальними компонентами: Запити (Queries, Q), Ключі (Keys, K), Значення (Values, V).

Q – репрезентація слова, на якому зараз сфокусована модель. На цьому етапі для кожного окремого слова послідовності створюється векторне представлення, яке відображає, наскільки це слово важливе в контексті всього тексту.

K – векторне представлення всієї текстової послідовності, враховуючи слово із Q. Матриця ключів дозволяє моделі зрозуміти, наскільки тісно пов'язане слово під фокусом із кожним із слів усієї послідовності.

Матриця V відповідає представленням сенсів слів. Ця матриця використовується в процесі для створення результуючої матриці шляхом її зважування за допомогою матриць K та Q.

Схематично процес створення результуючої матриці зображено на рисунку 3.3.

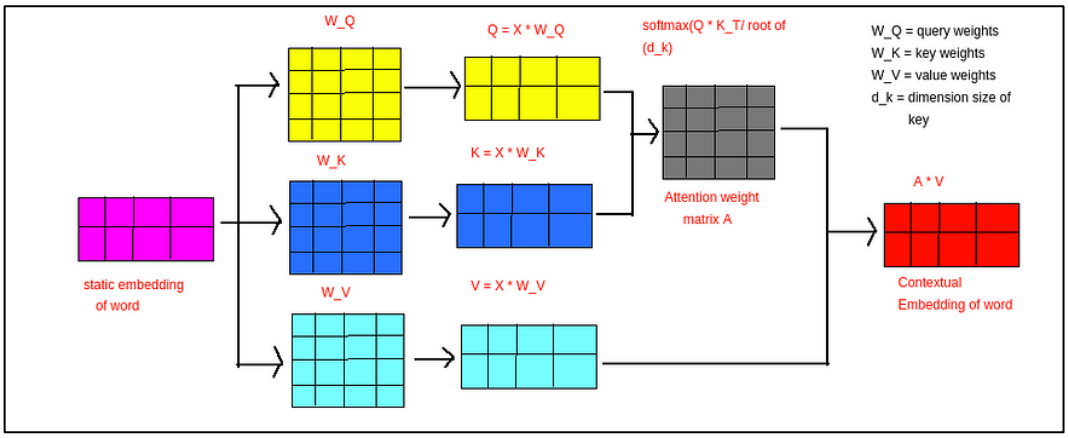

Рисунок 3.3 – Схематичне зображення процесу роботи механізму self-attention [19]

Розберемо детальніше процес створення цих матриць. Кожна з них утворена множенням відповідної матриці ваг (матриці W_Q, W_K, W_V відповідно) на початкову матрицю ембедингів, отриману після позиційного кодування, здійсненого на минулому етапі. Значення цих ваг підбирається під час процесу навчання моделі.

Наступним кроком є розрахунок матриці «Attention Score». Вона знаходиться як матричний добуток матриці Q та транспонованої матриці K. Результат представляє собою матрицю, яка відображає те, наскільки впливовим є певне слово для іншого в контексті всієї заданої текстової послідовності.

Після отримання цієї матриці до неї застосовується функція softmax. Ця функція приймає на вхід вектор і трансформує його значення відповідно до формули 3.7. Особливістю застосування функції softmax до матриці, а не до окремого вектора, є те, що вона застосовується незалежно до кожного рядка матриці.

Доречним є зауваження, що результат скалярного добутку зростає прямо пропорційно кількості вимірів. Оскільки матричний добуток фактично являє собою послідовність скалярних добутків векторів, це явище стосується й обчислення матриці Attention Score. У задачах NLP зазвичай оперують матрицями великого розміру, тому результати множення таких матриць можуть набувати дуже великих значень. Матриці ваг є навчуваними параметрами (learnable parameters), які оптимізуються за допомогою алгоритму зворотного розповсюдження помилки (back-propagation). Великі значення ваг на цьому етапі можуть спричиняти проблему вибухаючого градієнту. Для уникнення цієї проблеми застосовується масштабування шляхом ділення матриці Attention Score на квадратний корінь із кількості вимірів матриці K, що називається Scaled Attention Score.

Після виконання зазначених перетворень отриману матрицю Attention Matrix (A) використовують для матричного множення на матрицю V. В

результаті цих операцій утворюється зважена за контекстом матриця представлень слів, що використовується на наступному етапі обробки. Формальне вираження цього процесу наведено у формулі 3.3.

$$CE = AS \cdot V = softmax_{row}(SAS) \cdot V = \\ = softmax_{row}\left(\frac{AS}{\sqrt{d_k}}\right) \cdot V = softmax_{row}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V, \quad (3.3)$$

де *CE* – Contextual Embeddings;

V – матриця значень (Values);

AS – Attention Score;

SAS – Scaled Attention Score;

d_k – розмірність матриці ключів;

Q – матриця запитів (Queries);

K – матриця ключів (Keys).

Тепер, з розумінням роботи механізму Self-Attention, можна повернутися до Multi-Head Attention. Як було зазначено раніше, суть цього механізму полягає у створенні кількох «поглядів» на задачу. Це дає змогу кожній голові зосереджуватися на різних елементах послідовності. Досягнення цього забезпечується розділенням вхідної матриці на кілька підматриць зі зменшеною розмірністю. Важливо підкреслити, що скорочується саме розмірність ембедингів, а не кількість токенів. Тобто кожна голова отримує доступ до повної послідовності токенів, проте фокусується лише на тих аспектах, які входять до її «кола уваги». Наприклад, якщо на вхід подається матриця ембедингів розмірністю (10,512) і використовується 8 голів, то кожна голова опрацьовує матрицю розміром $(10,{}^{512}/_{8})$.

Після розділення вхідної матриці механізм працює за тими ж принципами, що й Self-Attention, проте формується не один набір матриць

Q, K та V, а по одному для кожної голови. Отримані результуючі матриці з усіх голів конкатенуються, утворюючи матрицю початкової розмірності.

Останній етап механізму полягає у знаходженні матриці W_0 (шляхом back-propagation) та її множенні на матрицю, отриману на попередньому кроці. Цей етап дозволяє головам узгодити, які елементи є найважливішими для фокусування уваги. Механізм Multi-Head Attention, який застосовується в енкодері, також відомий як «Encoder Attention».

Також важливим компонентом трансформерів є блок, підписаний на рисунку 3.2 як «Add & Norm». Цей блок реалізує механізм, відомий як Residual Connection (або Skip Connection), з додатковими особливостями. Термін «Skip Connection» точно відображає принцип роботи, який полягає у пропуску певного блоку моделі. Такий підхід спрямований на запобігання проблемі згасаючого градієнту під час навчання. На рисунку 3.4 наведено діаграму, що ілюструє цей принцип.

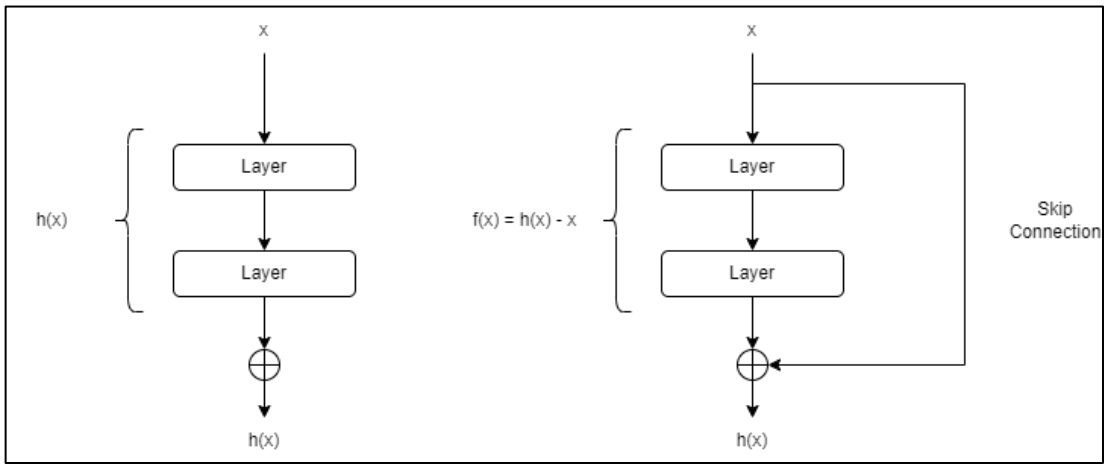

Рисунок 3.4 – Принцип Skip Connection [25]

Назва блоку Add & Norm точно відображає його функціональність: додати та нормалізувати. Тобто, до матриці, отриманої на попередньому кроці, додається матриця, яка подавалась на вхід попереднього шару. Після додавання здійснюється нормалізація, щоб зменшити вплив зміни розподілу

значень отриманої матриці на ваги. Для цього використовується Layer Normalization. Цей метод дозволяє нормалізувати всю матрицю незалежно від її розмірів і довжини вхідної послідовності. Проте, незважаючи на корисність, цей підхід має недоліки. Стискання значень до діапазону [0, 1] може бути надто жорстким, тому формулу доповнюють двома параметрами для навчання: γ (мультиплікативний) та β (адитивний). Ці параметри дають змогу моделі застосовувати нормалізацію більш гнучко. Тому формула нормалізації має такий вигляд:

$$\hat{x}_{m,i} = \gamma_i \cdot \frac{x_{m,i} - \mu_m}{\sqrt{\sigma_m^2 + \epsilon}} + \beta_i, \tag{3.4}$$

де $\hat{x}_{m,i}$ – нормалізоване значення елементу матриці;

γ_i – learnable параметр масштабування із індексом *i* свого вектора;

$x_{m,I}$ – ненормалізований елемент матриці:

μ_m – середнє значення вектора x_m;

β_i – learnable параметр зсуву із індексом *i* свого вектора;

σ_m^2 – значення дисперсії вектора x_m;

ϵ – дуже мала додатня константа задля запобігання ділення на нуль.

Наступним етапом трансформації даних є шар Feed Forward Neural Network (FFN). Якщо Attention-механізм забезпечує контекстну взаємодію між токенами в послідовності, то FFN здійснює незалежну нелінійну трансформацію для кожного токена окремо. Архітектурно цей компонент являє собою звичайну повнозв’язану нейронну мережу, що складається з трьох шарів: вхідного, прихованого та вихідного. Розмірність вхідного та вихідного шарів, як правило, становить 512, тоді як прихований шар має розмірність 2048. В якості активаційної функції між шарами використовується ReLU (Rectified Linear Unit). Формально, роботу FFN можна описати наступним виразом:

$$FFN(x) = RELU(xW_1 + b_1)W_2 + b_2 = max(0, xW_1 + b_1)\,W_2 + b_2, \quad (3.5)$$

де *FFN(x)* – результуюча матриця токену *x* після Feed Forward;

W_1 – матриця ваг між вхідним шаром та прихованим. Learnable параметр;

b_1 – біас між вхідним шаром та прихованим;

W_2 – матриця ваг між прихованим шаром і вихідним;

b_2 – біас між прихованим шаром та вихідним;

3.2.2 Будова decoder-only моделей

Decoder-only моделі, на відміну від encoder-only, складаються виключно з декодерної частини. Незважаючи на відсутність енкодера, вони здатні як інтерпретувати вхідний текст, так і генерувати відповіді, тобто можуть виконувати завдання типу text-to-text без окремого блоку кодування. У таких моделях розуміння контексту досягається завдяки модифікованому механізму self-attention, який включає каузальне маскування, що обмежує увагу лише на попередні токени послідовності. Проте, однією з відомих проблем цієї архітектури є деградація уваги (attention degradation), яка полягає у зниженні ефективності механізму уваги зі зростанням довжини вхідної послідовності. Незважаючи на цей недолік, моделі з архітектурою decoder-only активно застосовуються у завданнях генерації тексту, зокрема – написання історій, автодоповнення речень тощо. Найвідомішими представниками цього класу є моделі на основі GPT: GPT-1, GPT-2, GPT-3, GPT-4, ChatGPT.

Як і з encoder-only моделями структура моделі включає декілька шарів декодерів. GPT-1 має 12 шарів, GPT-2 – 48 шарів, GPT-3 – 96 шарів, GPT 4 – 96. На базі цих моделей є також збільшені та зменшені варіації із іншою кількістю шарів, які мають свої недоліки й переваги. Структуру decoder-only моделей наведено на рисунку 3.5.

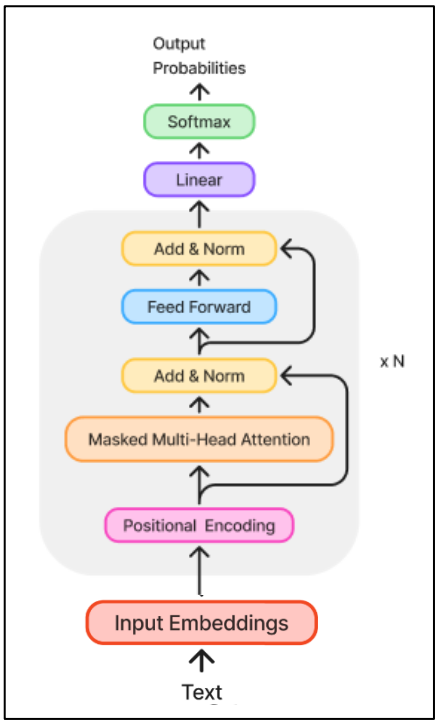

Рисунок 3.5 – Схематичне зображення decoder-only моделі

Як видно зі структури, декодерні моделі використовують замаскований (Masked) Multi-Head Attention. Ця модифікація обмежує здатність моделі фокусуватися на токенах, що розташовані після поточного у послідовності, тим самим забезпечуючи каузальність у процесі генерації тексту. Зазначений механізм є функціонально подібним до прийому Teacher Forcing, який полягає у подачі на вхід декодеру зсунутого праворуч еталонного виходу. Проте між цими двома підходами існує принципова відмінність: у разі використання Teacher Forcing модель не має доступу до поточного токена і повинна згенерувати його на основі попереднього контексту, тоді як у Masked Multi-Head Attention модель може бачити поточний токен і самостійно вирішує, наскільки важливо концентрувати увагу на ньому в контексті загального змісту послідовності.

Математично, це можна представити формулою 3.6.

$$Masked\ Scores = \begin{cases} QK^T/\sqrt{d_k}, & \text{якщо немає майбутніх токенів,} \\ -\infty & - \text{ в протилежному випадку.} \end{cases} \quad (3.6)$$

Тобто, як можна побачити, токенам, які не має бачити модель надається дуже мале від'ємне значення. Через це елементи матриці під час проходження через функцію softmax отримають дуже малі показники уваги і будуть дуже незначними за впливом на результат.

Оскільки ембединг-представлення не є придатним для прямого перетворення у вихідні токени, необхідним є їхнє перетворення у форму, з якої можна здійснити декодування. З цією метою використовується шар із лінійним перетворенням (Linear Layer), що виконує трансформацію матриці ембедингів, отриманої після проходження шару Feed Forward, у логітне представлення. Логіти утворюють матрицю, у якій кожен елемент відповідає певному слову з наперед визначеного словника. Це дозволяє моделі сформувати прогноз для всіх можливих токенів, а не лише тих, що були присутні у вхідній послідовності. Математично це зводиться до знаходження матриці ваг W, яка, будучи навчуваним параметром (learnable parameter), забезпечує відображення простору ембедингів у простір логітів.

Після утворення матриці логітів використовується функція softmax, яка є активаційною функцією перед вихідним шаром моделі.

Softmax, якщо говорити простою мовою – активаційна функція, яка повертає в якості результату ймовірнісний розподіл міток класів в проблемі багатокласової класифікації. Математично функція має такий вигляд :

$$softmax(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \tag{3.7}$$

де x – вектор чисел на нейронах;

N – кількість цих самих нейронів.

Ця функція має декілька властивостей: негативні значення будуть позитивними, значення лежать в діапазоні (0,1), в сумі всі значення

дорівнюватимуть 1. Тобто, як можна побачити, дана функція ідеально підходить для ймовірностей.

Саме Для цього її і використовують. Тобто, для визначення наступного слова формується вектор ймовірностей для кожного слова, який відображає шанс того, що певне слово буде наступним в послідовності. Логічно, що в якості кандидата для наступного слова обирається слово із найбільшим шансом.

3.2.3 Будова encoder-decoder моделей

Encoder-decoder моделі використовують обидва елементи – і енкодер і декодер. Такий підхід корисний у задачах, коли треба трансформувати один текст в інший, наприклад, задачі машинного перекладу. Як можна здогадатись, задача типу text-to-sql включає в себе елемент трансформації тексту природною мовою в SQL-запит, тому архітектура encoder-decoder є найкращім рішенням для вирішення поставленої в цій роботі задачі.

На вхід до декодерної частини подається ще і очікуваний результат моделі, проте із зсувом на 1 токен праворуч. Output Embeddings на вході задіяні лише під час навчання. Такий підхід називається Teacher Forcing. Цей метод дозволяє стабілізувати навчання моделі. Модель замість того, щоб робити прогноз наступного токену на основі свого попереднього результату, звіряється з правильною послідовністю. А зсув праворуч потрібен, щоб модель не «підглядала» токен який вона має передбачити. Тобто, модель бачить лише ті токени, правильної послідовності що вже мала згенерувати, але не поточний і не майбутній.

Схематичне зображення decoder-encoder архітектури наведено на рисунку 3.6.

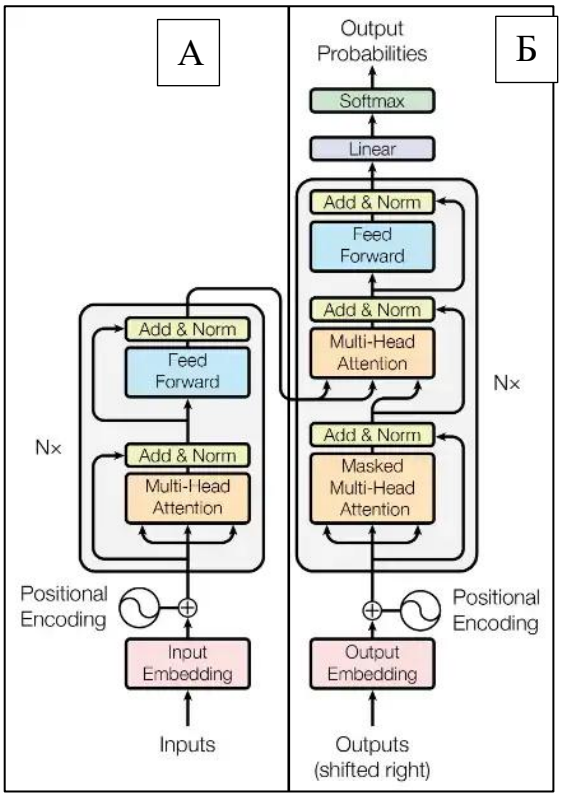

Рисунок 3.6 – Схематичне зображення encoder-decoder моделей [10]: А – енкодерна частина, Б – декодерна частина

У структурі декодера присутній додатковий блок Multi-Head Attention, розташований на стику з енкодером. Цей блок отримав назву «Encoder-Decoder Attention», оскільки виконує функцію з'єднання між енкодерною та декодерною частинами трансформера. Механізм реалізується таким чином: матриці ключів (K) та значень (V) формуються на основі виходу енкодера, тоді як матриця запитів (Q) формується декодером. Таким чином, декодер, використовуючи Q, здійснює запит на основі вже згенерованої послідовності, тоді як енкодер, використовуючи K і V, надає релевантну інформацію про вхідні дані, необхідну для генерації наступного токена.

Серед відомих представників моделей, побудованих на архітектурі типу encoder-decoder, можна виокремити MarianMT, Pegasus, BART, T5 тощо. MarianMT є модифікацією моделі BART, орієнтованою переважно на задачі машинного перекладу. Pegasus призначена для вирішення задач

узагальнення тексту (text summarization) і вирізняється спеціально адаптованим механізмом маскування вхідних даних, оптимізованим під специфіку цієї задачі. Модель BART є універсальним рішенням для широкого кола задач у межах підходу «text-to-text», таких як генерація, відновлення або трансформація тексту. Аналогічну функціональність має модель T5, яка, хоча й концептуально схожа на BART, має відмінну архітектуру переднавчання (pretraining) та є новішою розробкою. Враховуючи характер завдання, найбільш релевантними з точки зору ефективності є моделі BART і T5. У межах цієї роботи було прийнято рішення використати одну з їхніх переднавчених версій, доступних у бібліотеці transformers від компанії Hugging Face, яка забезпечує інтерфейс як для їхньої інтеграції, так і для донавчання (finetuning) відповідно до специфіки конкретної задачі.

BART (Bidirectional and Auto-Regressive Transformers) було створено з метою об’єднання технологій BERT та GPT. В результаті було отримано потужну модель, яка демонструє гарні результати у всіх text-to-text завданнях. На рисунку 3.7 наведено схематичне зображення внутрішньої структури BART.

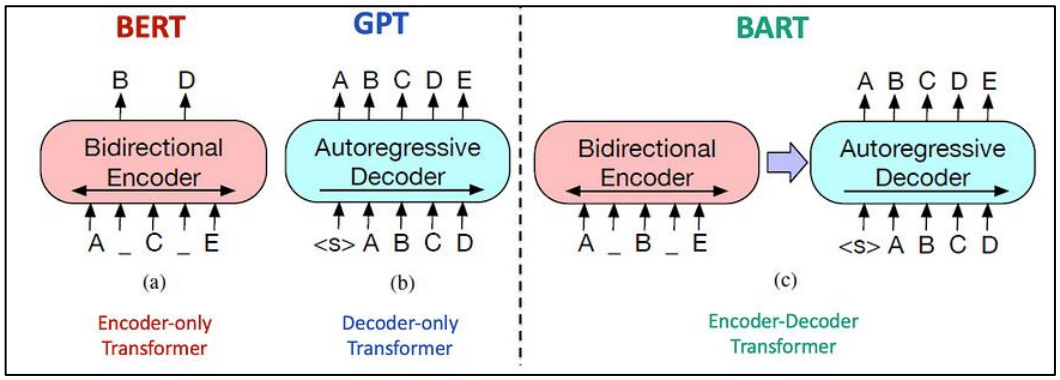

Рисунок 3.7 – Схематичне зображення внутрішньої структури моделі BART [15]

T5 (Text-to-Text Transfer Transformer) – інша потужна модель, яка призначена для вирішення text-to-text завдань. Основною відмінністю від BART є підхід до переднавчання моделі. Якщо BART намагається відтворити початкову послідовність з нуля токен за токеном, то T5 відновлює лише замасковані токени.

3.2.4 Висновки з вибору моделі

Хоча для генерації тексту зазвичай застосовуються моделі з різною архітектурою, зокрема decoder-only, як-от GPT (що стала основою для ChatGPT, розробленого компанією OpenAI), можливість генерації не є винятковою рисою лише таких моделей. Навіть моделі типу encoder-only, як BERT, за певних модифікацій архітектури також здатні виконувати завдання генерації тексту.

У межах даної роботи було вирішено зосередитися на моделях із архітектурою типу encoder-decoder, оскільки вони поєднують глибоке розуміння контексту завдяки енкодеру з ефективною генерацією тексту через декодер. У курсі, присвяченому великим мовним моделям, компанія Hugging Face рекомендує використовувати такі моделі цієї архітектури, як BART і T5 , зокрема для задач машинного перекладу [31]. Саме ці моделі було обрано для подальшого донавчання відповідно до поставленої задачі з метою визначення найоптимальнішої з них.

3.3 Вибір метрик оцінювання

Існує безліч різних моделей, які, теоретично, виконають поставлену задачу генерації SQL-запиту. Проте, яким чином зрозуміти, що якась модель справляється краще за іншу, а яка не дотягує? Через те, що генерація тексту – дуже специфічна і комплексна задача для неї неможна використовувати звичайні метрики оцінки, так і як, наприклад, процент

«вгаданих» слів у реченні (accuracy score), як це робиться в класичних моделях машинного навчання під час класифікації. Ці методи можуть підходити в специфічних завданнях, коли важлива точна постановка слів у реченні і недопустимі перефразування. Проте, зазвичай, одну і ту саму думку можна сказати декількома варіантами і вони також будуть правильними. В тому числі, неможна використовувати метрики оцінки для задач регресії.

Варто сказати, що метрики, під час навчання можуть відрізнятися від метрик під час кінцевої оцінки моделей. Наприклад, в трансформерах використовується Cross-Entropy Loss для оцінки моделі в ході навчання. Як і всі loss-функції модель намагається її мінімізувати під час оптимізації. Ця метрика оцінки використовує ймовірності і нагороджує модель за більші ймовірності для правильних результатів. Як і було сказано раніше, результат функції softmax, фактично, є представленням багатокласової класифікації. Проте, кількість класів у text-to-text завданнях відповідає кількості унікальних слів. Функція Cross-Entropy Loss для мультикласової класифікації має наступний вигляд:

$$Loss = -\sum_{k=1}^{K} y_k \log(p_k), \tag{3.8}$$

де y_k – очікувана ймовірність класу k, зазвичай, представлені як 1 для вірного класу і 0 для всіх інших (one-vs-all);

p_k – ймовірність, яку передбачила модель.

Проте, для вибору моделі використовують інші метрики. Найбільш відомими і уживаними серед них є ROUGE, BLEU, METEOR. Всі ці метрики недискретизовані, не диференційовані, або просто не мають стабільного градієнту. Через це, їх просто неможливо використовувати під час градієнтного спуску, за допомогою якого оптимізуються трансформери. Тому ці метрики відносяться до пост-оцінки моделі. Ми будемо

використовувати їх для порівняння моделей між собою. Проте, спершу потрібно з’ясувати яку метрику, або поєднання метрик ми маємо використовувати задня оцінки. Для цього детальніше розберемо їх.

3.3.1 BLEU

BLEU (Bilingual Evaluation Understudy) оцінює точність n-грам між згенерованою послідовністю та еталонною. В цій метриці передбачено штраф за занадто короткі речення. Посилаючись на класичні методи машинного навчання, можна сказати, що BLUE – це метрика precision в області оцінки текстових послідовностей. Проте, ця метрика має погану чутливість до семантики, через те, що вона порівнює дві послідовності, не допускаючи факту перефразувань.

Значення оцінки цієї метрики лежать в діапазоні [0, 1]. Де 1 – модель згенерувала послідовність, яка ідеально відповідає еталонній. Дана метрика, зазвичай, використовується для оцінки якості перекладів, проте, її використовують і для інших завдань. Зазвичай, вона використовується для корпусів слів із n-грамамами, не n рівне 4. Математично, ця метрика має таке представлення:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right), \tag{3.9}$$

$$BP = \begin{cases} 1 & \text{якщо } c > r, \\ \exp\left(1 - \frac{r}{c}\right) & - \text{ інакше,} \end{cases} \tag{3.10}$$

де p_n – precision для n-грам (розраховується відповідно до формули 3.12);

w_n – вектор ваг для n-грам (зазвичай $\frac{1}{N}$);

BP – Bravery Penalty, штраф за короткі відповіді;

r – довжина еталонного речення;

c – довжина згенерованого речення.

3.3.2 ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) – метрика для оцінки резюмування тексту, проте для машинного перекладу його також можна використовувати. ROUGE має декілька варіантів оцінки, серед них: ROUGE-N, ROUGE-L, ROUGE-S.

ROUGE-N, аналогічно до BLEU, порівнює n-грами еталонного тексту та згенерованого. Проте, не зважаючи на назву, ROUGE не рахує виключно recall, хоча в більшості випадків ця метрика використовується таким чином. За допомогою цієї метрики можна оцінити precision, recall та f-міру. Математично, формулу для розрахунку цієї метрики можна зобразити наступним чином:

$$ROUGE\text{-}N_{F\beta\text{-}score} = \frac{(1+\beta)^2 \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}, \tag{3.11}$$

$$Precision = \frac{C_R}{C}, \tag{3.12}$$

$$Recall = \frac{R_c}{R}, \tag{3.13}$$

де N – кількість токенів в одному n-грамі;

β – коефіцієнт важливості recall над precision;

C_R – кількість n-грам, зі згенерованої послідовності, які співпадають з n-грамамами із еталонної;

C – кількість всіх n-грам в згенерованій послідовності;

R_C – кількість n-грам із еталонної послідовності, що співпадають із згенерованою;

R – загальна кількість n-грам в еталонній послідовності.

ROUGE-L, в свою чергу, базується на найдовшій спільній підпослідовності. В такій послідовності слова можуть пропускати деякі слова, але все ще будуть йти по порядку написання. Наприклад, підпослідовність для послідовностей «The cat is on the mat.» та «The cat and the dog.» буде «the cat the». Математично f-міра ROUGE-L оцінки має такий вигляд:

$$ROUGE\text{-}L_{F\beta\text{-}score} = \frac{(1+\beta)^2 \cdot Precision\text{-}L \cdot Recall\text{-}L}{\beta^2 \cdot Precision\text{-}L + Recall\text{-}L}, \tag{3.14}$$

$$Precision\text{-}L = \frac{LCS}{C}, \tag{3.15}$$

$$Recall\text{-}L = \frac{LCS}{R}, \tag{3.16}$$

де LCS – longest common subsequence – довжина найбільшої спільної підпослідовності.

ROUGE-S працює аналогічно до ROUGE-N із N=2 (ROUGE-2) проте з деякими видозмінами. Справа в тому, що ROUGE-2 вважає за помилку формально правильні словосполучення якщо вони розділені словами. Таким чином, було запропоновано додати пропуск декількох слів між словами біграму. Таким чином, це збільшує кількість біграм для порівняння, проте разом з цим збільшується і об’єктивність оцінювання. Формули для цієї метрики відповідають формулам 3.11–3.13, проте із n = 2 та із взяттям біграм із пропуском слів.

До переваг метрик ROUGE можна віднести високу кореляцію з оцінками, що надаються людиною-експертом, що робить їх ефективним

інструментом для автоматизованої оцінки якості згенерованого тексту. Водночас до недоліків даних метрик належить відсутність урахування лексичної варіативності, зокрема синонімії, що може призводити до заниження оцінки текстів, які є семантично коректними, але сформульовані іншими словами.

3.3.3 METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) – метрика для оцінки якості машинного перекладу за допомогою n-грам. Ця метрика враховує синоніми та морфологічні варіанти слів.

Оцінка метрикою відбувається у декілька етапів. На першому етапі порівнюються уніграми (1-грами, або просто одиничні токени) згенерованого речення і еталонного. При чому, зіставлення виконують два різні модулі: точний (exact), стеммер Портера (Porter stemmer) та WN синонімізацію (WN Synonymy, WordNet Synonymy). Всі три модулі створюють пари слів між еталонним та згенерованим. Точний модуль знаходить слова, кожна літера яких співпадає. Стеммер Портера знаходить слова, обрізаючи закінчення. Тобто, слова «computer» та «computers» обріжуться до «comput» і будуть вважатись одним словом. WN синонімізатор буде перевіряти на синоніми слів, утворюючи пари, на кшталт computer – machine.

На другому етапі пари із кожного модулю об’єднуються в одну множину. Задачею на цьому етапі є знайти однозначне вирівнювання між еталонним реченням та згенерованим. Тобто кожне слово може відповідати максимум одному слову в іншій послідовності. Після виконання цієї умови обирається підмножина із найбільшою кількістю елементів у ній. Якщо декілька множин мають однакову потужність – обирається та, яка має менше всього «перетинів» між словами. Простими словами перетин слів можна пояснити наступним чином. Візьмемо обидві послідовності і

напишемо, скажімо, еталонну над згенерованою. Після чого з'єднаємо пари із обраної множини. Якщо лінії перетнулись – це і є перетин. Таких перетинів має бути мінімальна кількість. Математично перевірка на перетин слів виглядає так:

$$(pos(t_i) - pos(t_k)) \cdot (pos(r_j) - pos(r_l)) < 0 \Rightarrow \text{перетин}, \quad (3.17)$$

де *pos(*t_i*), pos(*t_k*)* – порядковий номер слів t_i та t_k згенерованого речення;

pos(r_j*), pos(*r_l*)* – порядковий номер слів r_j та r_l еталонного речення.

Наступним етапом йде розрахунок precision та recall, аналогічно до формул 3.15 та 3.16, відповідно, але замість довжини найбільшої спільно підпослідовності (LCS) буде використано потужність знайденої підмножини пар токенів. Після чого відбувається розрахунок f-міри за наступною формулою:

$$METEOR\text{-}F = \frac{10PR}{R + 9P}, \quad (3.18)$$

де P – precision;

R – recall.

Як можна побачити, у метриці METEOR більша вага надається компоненті recall, що дозволяє краще враховувати повноту збігів. Водночас базова формула не враховує семантичну подібність, оскільки використовує лише уніграми під час обчислень.

Для часткового врахування структури та послідовності правильних збігів у введенні використовується спеціальний механізм – групування збігів у фрагменти (chunks). Кожен фрагмент включає тільки ті слова, які зіставлені коректно і розташовані у відповідному порядку як у згенерованій, так і в еталонній послідовності.

У найкращому випадку, коли вся послідовність є правильною і впорядкованою, буде лише один фрагмент. У найгіршому – кожен збіг буде

окремим фрагментом, тобто їхня кількість дорівнюватиме числу уніграмних відповідностей. Штраф для кількості фрагментів розраховується за наступною формулою:

$$Penalty = 0.5 \cdot \left(\frac{chunks}{unigrams}\right)^3, \quad (3.19)$$

де *chunks* – кількість фрагментів;

unigram – кількість уніграм в підмножині.

Фінальна формула для розрахунку метрики METEOR має такий вигляд:

$$METEOR = METEOR\text{-}F \cdot (1 - Penalty), \quad (3.20)$$

3.3.4 Висновки з вибору метрик

В роботі розглядається генерація коду – особливий випадок машинного перекладу, проте зі строгими синтаксичними вимогами. Мови програмування не допускають креативності чи синонімів. Через це метрики, які заохочують перефразування (наприклад, METEOR), нам не підходять.

Крім того, важливо уникати зайвих слів у згенерованому коді, оскільки це може спричинити синтаксичні чи логічні помилки. Recall-орієнтовані метрики не штрафують за надмірну генерацію, тому вони непридатні. Зокрема, ROUGE-S не враховує порядок слів, що є критичним для коду. ROUGE-N та ROUGE-L могли б бути корисними при низьких β, але BLEU буде кращім рішенням. Вона враховує precision і вводить штраф за короткі відповіді, стимулюючи моделі генерувати коректні послідовності оптимальної довжини. Тому для оцінки було обрано саме BLEU. Як було сказано в розділі 3.3.1 за замовчуванням використовують BLEU-4, проте для речень це надто суворо, тому було вирішено використати BLEU-2.

4 ПРАКТИЧНІ ДОСЛІДЖЕННЯ ТА РОЗРОБКА ЗАСТОСУНКУ

4.1 Запропоновані методи покращення навчання моделі

Задля збагачення текстових запитів, які подаються на вхід моделі було вирішено додати перефразування текстових запитів. Для цього завдання було вирішено обрати відповідну модель.

Крім цього, важливо також щоб результати перефразувань були доволі креативними. Тому що додавання рядків, в яких відсутнє, або додано всього одне слово майже не вплинуть на результат.

Саме в якості моделі для перефразувань було вирішено обрати humarin/chatgpt_paraphraser_on_T5_base [32]. Ця модель добре себе проявляє в перефразуванні в різних стилях.

На рисунку 4.1 представлено приклади перефразувань цієї моделі.

```
What is the total trade value and average price for each trader and stock in the trade_history table? ->
 -> In the trade_history table, what is the trade value and average price for each individual trader and stock?

Find the energy efficiency upgrades with the highest cost and their types. ->
 -> Identify the most expensive energy efficiency upgrades and their corresponding amounts.

What is the total spending on humanitarian assistance by the European Union in the last 3 years? ->
 -> What is the amount of money that the European Union has spent on humanitarian aid in the past three years?

What is the average water temperature for each fish species in February? ->
 -> What is the typical water temperature for fish in February?

Delete a program's outcome data ->
 -> Eliminate a program's output from the output data.
```

Рисунок 4.1 – Декілька прикладів перефразувань моделі chatgpt_paraphraser_on_T5_base

Як можна побачити, результат дуже гарний. Усі речення дуже сильно відрізняються від оригінальних варіантів. Проте, неможна бути впевненим в тому, що всі 100,000 речень буде перефразовано коректно, деякі речення можуть дублюватися, а деякі можуть дуже сильно змінити сенс оригіналу.

Це створить у моделі неправильне представлення про правильні послідовності і створить аномальні значення у моделі на деяких токенах.

Тому було вирішено додати перевірку якості перефразувань за допомогою METEOR. Дана метрика враховує як стеммінг, так і синоніми слів, завдяки чому модель не буде штрафуватись за додавання синонімів у перефразування. Було вирішено обмежити діапазон допустимого значення для METEOR, розрахувавши медіанне значення цієї метрики для 100 перефразувань. Верхнею межею буде значення на 40% вище за медіанне, а нижньою – на 40% нижчою. Це дозволить прикладам бути креативними, але і зберегти сенс. Медіанне значення не так сильно піддається впливу викидів, тому було вирішено використовувати його замість середнього.

Проте, генерація 100 прикладів разом із підрахунком метрик зайняла 86 секунд. Тобто, генерація і перевірка всіх прикладів буде відбуватись приблизно одну добу, тому було прийнято рішення оптимізувати процес навчання. Для оптимізації перефразувань було вирішено перенести їх створення на GPU (Graphical Processor Unit) у середовищі Google Collab. Той самий об’єм перефразувань було створено за 2 секунду. Тому очікуваний час створення 100,000 перефразувань – близько 30 хвилин. Це вже є дуже прийнятним результатом.

Після обробки було отримано близько 84,000 перефразувань, що доповнять навчання.

Трансформер, як і майже будь яка модель із області машинного навчання, має гіперпараметри. Гіперпараметри поділяються на архітектурні, генеративні і тренувальні. Серед архітектурних – кількість енкодер або декодер шарів, розмірність моделі, розмір словнику тощо. Ці гіперпараметри актуальні лише задачі переднавчання моделі. Якщо ми виконуємо донавчання – архітектурні параметри вже визначено і їх, не змінюють. Генеративні – параметри, які відповідають за якість генерації. Наскільки модель креативна, або стримана, наскільки великі послідовності може генерувати. Тренувальні гіперпараметри контролюють процес

навчання моделі. Вони прямим чином впливають на якість навчання моделі тому їх також треба оптимізувати.

Серед основних тренувальних параметрів можна виділити learning_rate – він впливає на швидкість збіжності моделі. Він використовується під час оновлення ваг і визначає наскільки сильно ці ваги змінюються. Формула 4.1 відображає вплив цього гіперпараметру на процес обчислення майбутніх значень ваг.

$$\theta_{t+1} = \theta_t - \alpha + \frac{\partial L}{\partial \theta_t}, \tag{4.1}$$

де θ_t та θ_{t+1} – ваги попередня і поточна, відповідно;

α – learning_rate;

$\frac{\partial L}{\partial \theta_t}$ – градієнт функції витрат L відносно ваги θ_t.

Number_of_epoch також є доволі важливим параметром. Він відповідає за те, скільки разів модель пройде через тренувальний датасет під час навчання.

Також значним гіперпараметром є warmup_ratio. На початку навчання модель «розігрівається» – починає з малих значень learning_rate і поступово збільшує його до вказаного перед початком навчання. Це дозволяє стабілізувати навчання на початку. Цей гіперпараметр визначає частку загальної кількості кроків навчання, протягом яких відбувається розігрів

Параметр weight_decay відповідає за L2 регуляризацію. Тобто, штраф за занадто складну функцію із великими значеннями ваг, що перешкоджає перенавчанню. L2 регуляризація визначається як сума квадратів ваг моделі, який додається до функції втрат, таким чином моделі просто невигідно переускладнювати функцію.

Batch_size відповідає кількості прикладів, які обробляються моделлю одночасно перед оновленням ваг. Великі значення прискорюють процес

навчання, малі – сповільнюють. Проте, на великих значеннях модель гірше узагальнює і може застрягти в локальному мінімумі.

4.2 Навчання моделей

Варто зазначити, що в датасеті присутні дві колонки для генерації: сам запит англійською і контекст (схема) таблиці. Задавати дві колонки для навчання моделі не є прийнятною практикою, тому колонки конкатенуються, утворюючи цільний текстовий рядок із роздільником «|»

Від початку, перевіримо гіпотезу щодо того, що навчання із використанням перефразувань покращить результат. Для цього донанавчимо дві моделі BART: одну з перефразуваннями, іншу без. І порівняємо результати оцінки. Якщо результати задовільняють, то із великою імовірністю результат буде гарний і на інших моделях, наприклад, T5. Це дозволить не навчати T5 (яка є більш потужною і довше навчається) на датасеті який не покращує навчання.

Спочатку проаналізуємо середні і медіанні значення результатів. Викиди результатах мають доволі сильний вплив на середнє арифметичне значення, на відміну від медіани результатів. Таким чином можна оцінити куди зміщено розподіл: якщо середнє менше за медіану – дані мають багато аномально низьких значень, якщо медіана менше – навпаки.

Для зручності BART із перефразуваннями буде називатись «ParaBART» а без – «RawBART». ParaBART має значення медіани рівне 0.703, RawBART – 0.672. В свою чергу середні значення для ParaBART та RawBART – 0.686 та 0.664, відповідно. Медіанне значення більше за середнє, в обох випадках, що свідчить про велику кількість дуже низьких значень на окремих прикладах. Проте, ця ситуація відбувається в обох моделях, тому можна сказати, що самі перефразування в цьому не винні. Варто зазначити, що для цієї оцінки було використано лише тестові дані, тобто модель їх не бачила під час навчання.

Для перевірки на перенавчання варто порівняти результати на тестовому наборі із результатами на тренувальному. Медіани для RawBART та ParaBART на тренувальному наборі – 0.666 та 0.701, а середні – 0.660 та 0.684, відповідно. Результати на тестовому наборі та результати на тренувальному наборі не дуже різняться, тому, можна зробити висновок, що перенавчання немає в обох випадках.

Далі буде детальніше розібрано розподіл якості генерації. Схематично розподіл зображено на рисунку 4.2.

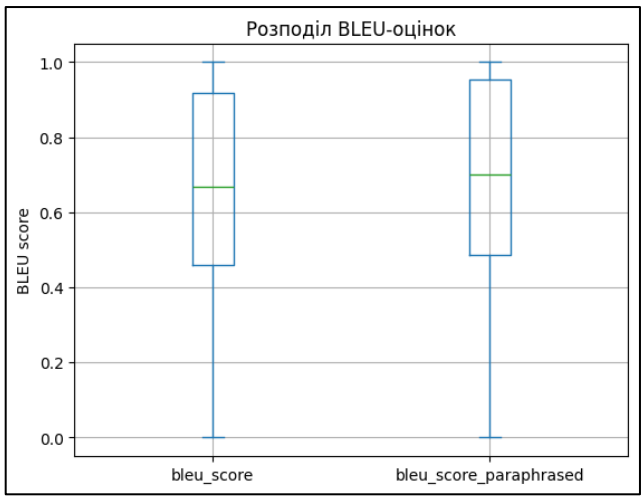

Рисунок 4.2 – Коробкові діаграми розподілу значень BLEU

На рисунку 4.2 можна побачити, що коробка діаграми ParaBART зміщена вище за RawBART, що означає, що оцінки моделі із перефразуваннями в цілому краще. В той же час, вуса обох моделей розміщено однаково і вони приймають значення 0 в мінімумі і 1 в максимумі. Це підтверджує гіпотезу про наявність викидів. Модель просто оцінює деякі значення аномально погано.

В розділі 1.3 було висунуто гіпотезу, що незбалансованість класів може спричинити погіршення результатів: класи із малою кількістю прикладів в тренувальному наборі рідко попадатимуть до моделі тому вона погано вчиться. Розподіл результатів оцінки зображено на рисунку 4.3.

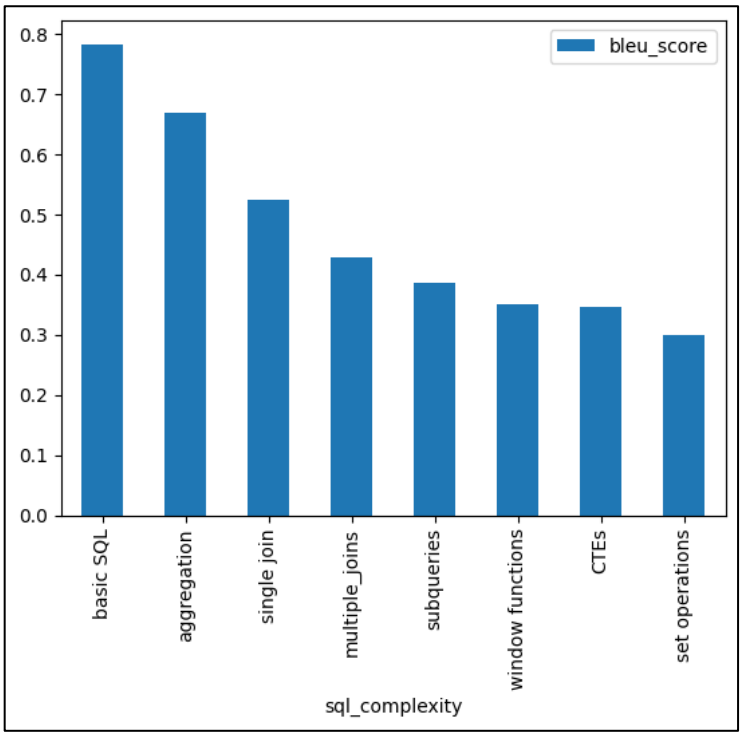

Рисунок 4.3 – Розподіл результатів оцінки ParaBART на тестовій виборці, згруповані за колонкою sql_complexity)

Візуально на рисунку 4.3 спостерігається деяка схожість розподілів із рисунком 1.4. Для об’єктивності було проведено кореляційний аналіз між розподілом класів колонки sql_complexity в тренувальній вибірці та значення оцінки на тестовій вибірці, згрупованій за колонкою sql_complexity. Кореляційний аналіз показав силу кореляції в 0.94 (для обох моделей приблизно однаково). Велика позитивна кореляція підтверджує факт прямої залежності між кількістю екземплярів класу в тренувальному наборі і його середньою оцінкою. Тобто, простіше кажучи, чим менше екземплярів тим гірша оцінка. Справа в тому, що як і було сказано в розділі 1.3, клас basic SQL перевершує своєю чисельністю інші класи в сумі. Проте, і в використанні запити цього класу найуживаніші. Тому, підлаштовуватись під малочисельні класи, які зустрічаються рідко було визначено неефективним.

В підсумку варто зазначити, що гіпотезу про те, що додавання перефразувань збільшить ефективність навчання було підтверджено і тому під час оптимізації гіперпараметрів буде використовуватися саме датасет із перефразуваннями.

Підбір гіперпараметрів було виконано за допомогою фреймворку optuna, який як раз спеціалізується на підборі гіперпараметрів. Цей фреймворк ефективно підбирає оптимальні значення за допомогою вбудованих функцій. Наприклад, стратегією за замовчуванням є TPESampler. Ця стратегія використовує Баєсівську оптимізацію. Вона знаходить оптимум за допомогою функцій щільності ймовірностей. Це дозволяє моделі не вчитись із гіперпараметрами, які, ймовірно, не принесуть покращення. Це набагато корисніше ніж GridSearch, тим паче в контексті глибоких нейронних мереж, моделі яких можуть вчитись декілька діб.

Через брак розрахункових потужностей, було вирішено провести лише 6 ітерацій навчання. Для підбору оптимального рішення для BART було сформовано такі масиви значень: діапазон learning_rate в межах $[1 \cdot 10^{-5}, 5 \cdot 10^{-4}]$ із логарифмічним кроком, weight_decay в межах [0, 0.3], warmup_ratio в межах [0, 0.3].

Після виконання підбору параметрів найкращій результат показала модель із такими значеннями: learning_rate – $3.73 \cdot 10^{-5}$, weight_decay 0.244, warmup_ratio – 0.25. Після цього було навчено модель з цими параметрами і оцінено за обраною метрикою.

Аналогічно BART для моделі T5 також було виконано пошук оптимальних гіперпараметрів. Найкращими значеннями стали: learning_rate – 0.005, weight_decay – 0.067, warmup_ratio – 0.199.

Варто зазначити, що через обмеження в розрахункових можливостях підбір гіперпараметрів виконувався із використанням для BART (bart-base) – повного датасету, але без перефразувань із кількістю епох, що дорівнює 2. Для моделі T5 (t5-small), в свою чергу, підбирала параметри на зменшеному вдвічі датасеті без перефразувань. Для неї також було

використано batch_size, що дорівнює 32 (для BART – 64). Оптимізація в обох випадках виконувалась протягом 4 годин із використанням усієї пам’яті графічного процесора.

Також важливо сказати, що навчання перед фінальною оцінкою для обох моделей виконувалось із використанням датасету із перефразуваннями, так як його було визначено як той, що покращує навчання.

Для визначення оптимальної моделі було виконано навчання із підрахунком метрики на тестовому датасеті і розрахунком медіанного і середнього значення усього тестового набору даних. Медіанне значення для BART – 0.749, а середнє – 0.721. В свою чергу середнє та медіанне для T5 – 0.726 та 0.762. Візуально всі навчені моделі зображено на рисунку 4.4.

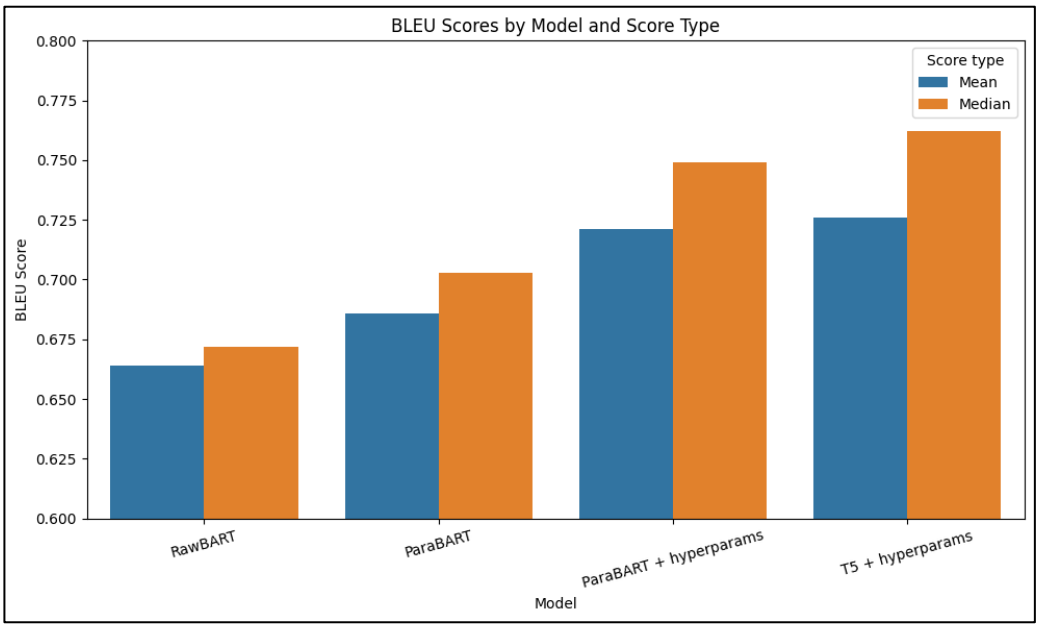

Рисунок 4.4 – Діаграма усіх моделей, що було навчено під час роботи

З цього можна зробити висновок, що для генерації sql-запитів в застосунку буде використано саме T5, яка навчена із застосуванням оптимальних гіперпараметрів та на датасеті із перефразуваннями.

4.3 Дослідження результатів оцінки T5

Для детальної оцінки якості навчання моделі було вирішено подивитись на медіанну оцінку за класами. Значення, отримані групуванням колонки sql_complexity зображено на рисунку 4.5.

sql_complexity	bleu_score
basic SQL	0.929320
aggregation	0.738067
single join	0.645122
multiple_joins	0.613139
subqueries	0.477459
window functions	0.437235
CTEs	0.367069
set operations	0.358569

Рисунок 4.5 – Медіанні значення результатів оцінки на тестовому наборі моделі T5, згруповані за колонкою sql_complexity

На рисунку 4.5 знову простежується проблема незбалансованості датасету, що може впливати на загальну якість моделі. Водночас, на даному зображенні чітко видно, чому медіанні значення за всіма оцінками перевищують середні: найбільший за кількістю прикладів клас – «basic SQL» – має оцінку майже 0.92. Такий результат є досить високим і свідчить про ефективність моделі у вирішенні завдань базового рівня. Це відповідає цільовому призначенню розроблюваного застосунку, який орієнтований на підтримку користувачів у виконанні простих, рутинних задач. Хоча здатність розв’язувати більш складні завдання є бажаною, вона не є критично необхідною для основного функціоналу системи.

Розподіл медіанних значень подано на рисунку 4.6 у вигляді коробкової діаграми.

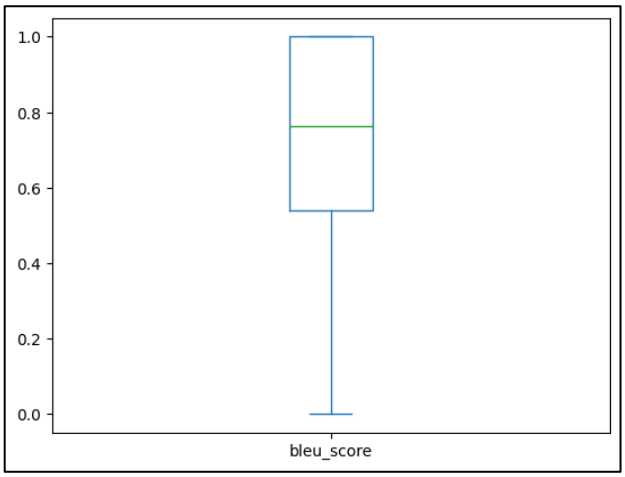

Рисунок 4.6 – Коробкова діаграма медіан оцінки моделі T5 на тестовому датасеті

На рисунку 4.6 також спостерігається дуже обрий результат. Коробка знаходиться доволі високо, а край третього квартиля співпадає з максимумом, проте нижній вус все ще лежить на значенні 0. Це означає, що модель в цілому генерує досить добре, але іноді дуже критично помиляється, генеруючи зовсім не те.

Для ще більш детального розуміння розподілу оцінки було побудовано гістограму, яку зображено на рисунку 4.7.

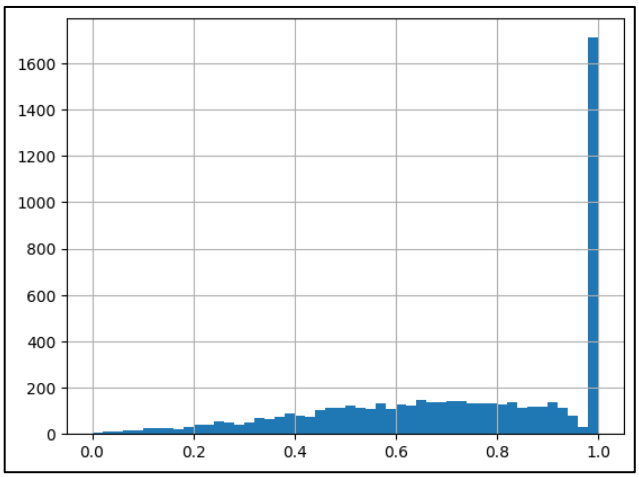

Рисунок 4.7 – Гістограма оцінок T5 на тестовому наборі

Можна побачити великий шпик біля значень, рівних одиниці. Це свідчить про те, що модель дуже велику кількість прикладів класифікує ідеально. Через те, що оцінка за допомогою BLEU відбувається із використанням біграм, такий результат свідчить про те, що абсолютно всі біграми співпали. Було вирішено детальніше проаналізувати аномально гарні значення.

Ідеальних оцінок метрикою – 1709, це 29% тестового набору, з них basic SQL – 23%. Проте, така оцінка не дуже об’єктивна, зважаючи на незбалансованість класів. Але, проаналізувавши ідеальні входження із врахуванням кількості екземплярів, можна побачити ще більш явну аномалію. На рисунку 4.8 зображено відсотки входжень, які мають значення оцінки 1 окремо серед свого класу. NaN помічено ті класи, які не мають жодного ідеального входження.

```
sql_complexity
basic SQL             0.474381
aggregation           0.203221
single join           0.065463
subqueries            0.054404
set operations        0.039216
multiple_joins        0.014388
CTEs                       NaN
window functions           NaN
Name: count, dtype: float64
```

Рисунок 4.8 – Відсотки ідеально згенерованих речень окремо за класами

Тут ще більш явно видно аномалію: майже половина класу basic SQL згенеровано ідеально. Дослідимо причину цього явища. Є ймовірність, що дане явище спричинено тим, що цей клас має надто прості входження і модель просто дуже добре зрозуміла шаблон. Для перевірки цієї теорії було вирішено розбити послідовності на слова і порахувати їх чисельність. Якщо

слів менше – запит простіший і тому оцінка буде кращою. На рисунку 4.9 зображено значення середніх довжин запитів, поділених на слова за кожним із класів колонки sql_complexity.

```
sql_complexity
basic SQL           11.275706
aggregation         15.103528
single join         20.931151
CTEs                21.000000
subqueries          24.077720
window functions    24.668367
set operations      26.431373
multiple_joins      32.417266
Name: len_split, dtype: float64
```

Рисунок 4.9 – Значення середніх довжин запитів, поділених на слова за кожним із класів колонки sql_complexity

Навіть неозброєним оком можна побачити деяку залежність між значеннями, зображеними на рисунку 4.9 та 4.8. Проте, для об’єктивності було проведено кореляційний аналіз, результатом якого є -0.87. Це значення свідчить про суттєвий зв’язок між досліджуваними змінними. Що можна вважати доволі сильною від’ємною кореляцією між відповідними даними. Простими словами: це доводить те, що чим менше в запиті слів, тим частіше модель буде генерувати ідеальну відповідь.

В підсумку можна сказати, що таким чином було підтверджено гіпотезу про те, що стрибок ідеальних значень обумовлено простотою запитів найбільшого з класів. Звісно, треба також врахувати те, що класи були дуже незбалансовані, що також вплинуло на загальну якість моделі. Такий дисбаланс є важливим фактором, який варто враховувати при інтерпретації результатів.

4.4 Створення інтерфейсу застосунку

Для створення інтерфейсу було вирішено використати фреймворк Next.js [34], який надає інструменти для зручного створення та адміністрування браузерних застосунків.

Проаналізувавши вимоги, наведені в розділі 2, було сформовано схематичну версію застосунку. Він буде складатись із двох сторінок: головної та сторінки визначення схеми. На рисунку 4.10 наведено схематичне зображення головної сторінки інтерфейсу застосунку.

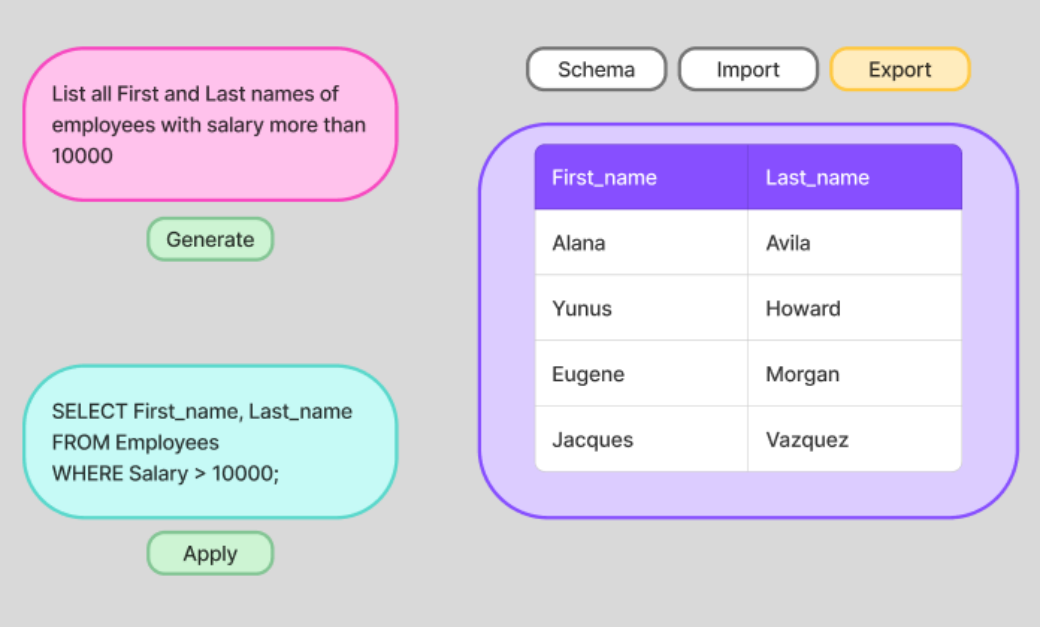

Рисунок 4.10 – Схематичне зображення головної сторінки інтерфейсу

Було вирішено розділити головну сторінку на два логічні блоки: блок роботи із запитами і блок роботи із таблицями. Блок роботи із запитами розміщено ліворуч і він включає в себе текстове поле для вводу тексту англійською мовою (рожеве), кнопки «Generate», текстового поля для введення SQL-запиту (блакитне) та кнопки «Apply» для виконання запиту. Саме в блакитному полі буде виводитись результат, який згенеровано моделлю, за допомогою запиту, який користувач увів у рожеве поле. Не

важно здогадатись, що кнопка «Generate» відправляє запит на генерацію до моделі, а кнопка «Apply» застосовує цей запит до БД.

В правому блоці розміщено три кнопки: «Schema», «Import», «Export» та таблицю. Кнопка «Import» має відкрити поле для вибору файлу для завантаження у застосунок, кнопка «Export» має зберегти результат який бачить користувач у таблиці, кнопка «Schema» має відкрити сторінку редагування, або створення схеми БД. На рисунку 4.11 схематично зображено вигляд сторінки редагування схеми.

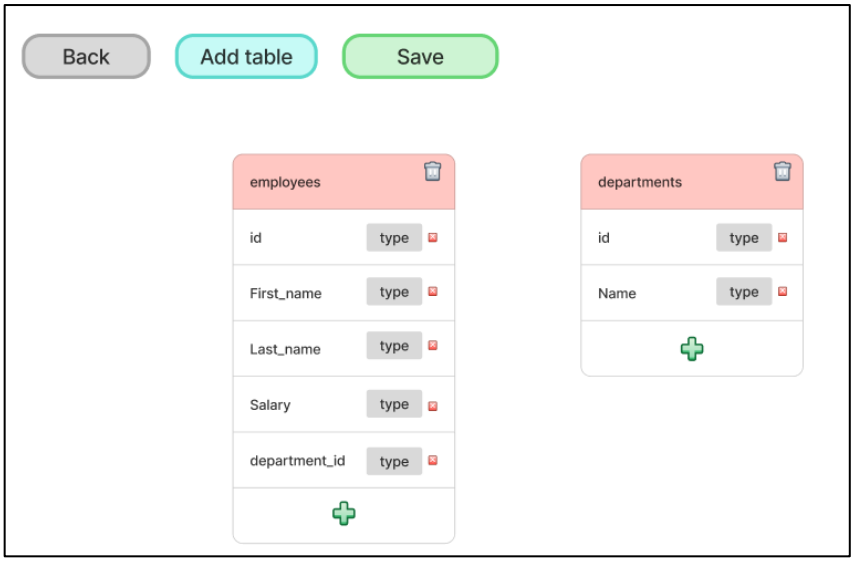

Рисунок 4.11 – Схематичне зображення сторінки для редагування схеми БД

Схема має панель керування, що розміщена зверху та поле, в якому зберігається сама схема. Серед кнопок панелі керування є: кнопка «Back», яка повертає на головну сторінку, кнопка «Add table», яка створює на сторінці нову таблицю і кнопка «Save» для збереження таблиці.

За допомогою подвійного кліку по назві колонки, або назві таблиці можна змінювати назву. Видалити колонку можна за допомогою хрестика біля колонки, додати за допомогою знаку «+» знизу. Видалення таблиці відбувається через натискання на смітничок біля назви таблиці. Вказання

типів даних таблиці відбувається через натискання на випадаючий список біля назви колонки.

Фінальний вигляд розробленої головної сторінки інтерфейсу зображено на рисунку 4.12.

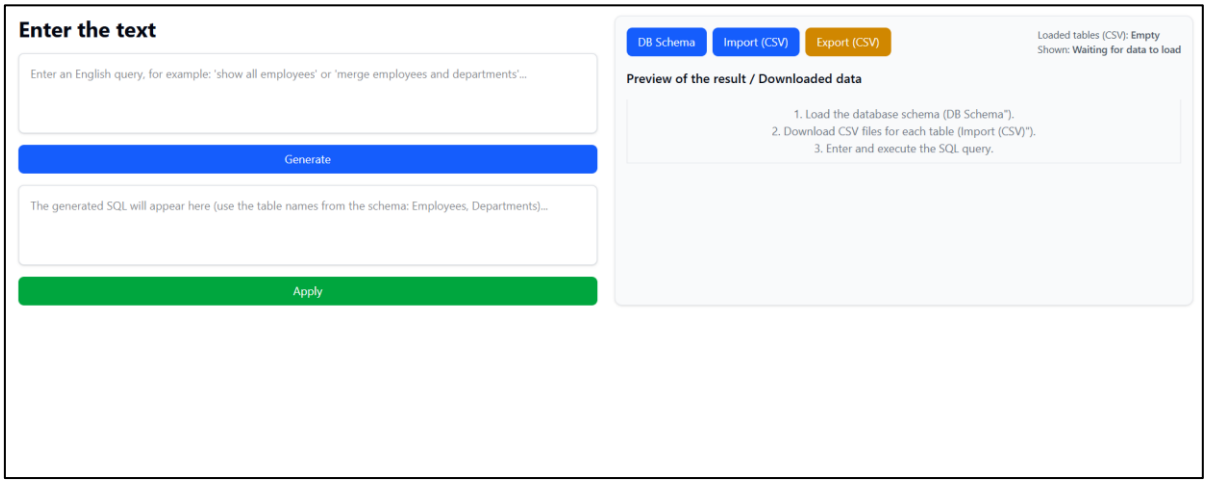

Рисунок 4.12 – Головна сторінка інтерфейсу

Було додано можливість додавати таблиці за допомогою CSV-файлів. Передбачено, що користувач спочатку створить потрібну йому схему і вже потім буде додавати дані, про що йому сповіщає підказка на місці таблиці передпоказу. Додати дані можна після натискання кнопки «Import (CSV)». Після натискання відкриється стандартний файловий провідник операційної системи і вікно повідомлення із проханням вказати назву для таблиці щоб правильним чином ця таблиця була асоційована зі схемою.

Після натискання кнопки «Export (CSV)» з’явиться контекстне вікно із проханням ввести назву для файлу перед його збереженням. Після підтвердження файл буде збережено у форматі CSV.

Після натискання кнопки «DB Schema» відкриється сторінка для роботи із схемою БД, фінальне зображення якої наведено на рисунку 4.13.

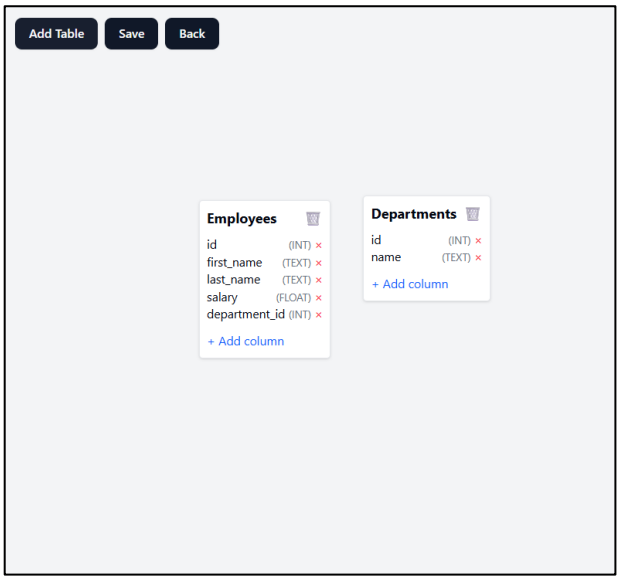

Рисунок 4.13 – Фінальний вигляд сторінки для роботи зі схемою БД

Будь-які збережені зміни в схемі записуються в локальне сховище браузеру (Local Storage) і, відповідно, звідти підвантаюжуться коли вони потрібні, що дозволяє не створювати схему кожного разу з нуля.

Знизу від кнопок можна побачити таблицю для передпоказу запиту. Ця таблиця показує лише перші 30 входжень таблиці щоб запобігти проблемам, пов’язаним із великими обсягами даних.

Після натискання на кнопку «Generate» схема (якщо є), завантажується з локального сховища, після чого до неї конкатенуються дані із таблиці (якщо є) і запит англійською мовою, відповідно до того, яким рядок виглядав під час тренування. Коли обробку завершено послідовність направляється на локальний сервер, написаний за допомогою фреймворку Flask на Python. Запит обробляється моделлю, яка повертає результат назад на клієнтську сторону застосунку в поле, де передбачається робота з SQL-запитом.

Після натискання «Apply» запит із відповідного текстового поля виконується на базі даних, яку завантажив користувач із виводом результату запиту в таблицю передпоказу. Звісно, якщо запит не видав результату (наприклад, додавання рядків) користувача буде попереджено.

ВИСНОВКИ

У процесі виконання роботи було проведено комплексний аналіз обраного набору даних, включно з детальним вивченням кожної колонки для визначення її потенційного впливу на ефективність навчання моделей.

Було ретельно досліджено архітектурні особливості моделей типів encoder-only, decoder-only та encoder-decoder. Описано їх базову структуру та механізми агрегації текстової інформації для генеративних задач, здійснено порівняльний аналіз спільних і відмінних рис цих архітектур із врахуванням їх сильних та слабких сторін. На основі цього аналізу обґрунтовано вибір найбільш доцільної архітектури для реалізації поставленої задачі.

Окрему увагу приділено вивченню ключових метрик оцінювання якості генеративних моделей, зокрема ROUGE, BLEU і METEOR. Вибір конкретних метрик було аргументовано з урахуванням особливостей предметної області. Визначено, що оптимальною для оцінювання є метрика BLEU-2.

У рамках роботи досліджено гіпотезу щодо підвищення якості генерації за рахунок використання перефразувань, створених іншою нейронною мережею. Обґрунтовано критерії та метрику оцінки креативності перефразувань, що дозволяє визначати порогові значення для оцінки їх адекватності.

Для оцінки креативності перефразувань застосовано метрику ROUGE-2 з встановленими межами від 0.6 до 1.4 відносно середнього значення за валідаційною вибіркою.

Здійснено вибір моделей для донавчання з урахуванням їх характеристик і потенціалу, виконано оптимізацію гіперпараметрів та проведено оцінку їх продуктивності. На основі результатів порівняння вибрано модель, яка продемонструвала найвищі показники відповідно до визначених метрик.

Модель BART показала кращі результати на датасеті з перефразуваннями, що стало підставою для застосування цього датасету у процесі навчання обох моделей.

Після проведення оптимізації гіперпараметрів моделей BART (bart-base) та T5 (t5-small) встановлено, що модель T5 забезпечує вищу продуктивність за обраною метрикою. Аналіз результатів виявив аномально високі значення для окремих класів, які, за допомогою кореляційного аналізу, було пов’язано із нерівномірністю розподілу класів та шаблонністю найбільш чисельного класу.

Особливий акцент зроблено на глибокому аналізі отриманих результатів, зокрема за допомогою аналітичного та кореляційного підходів для більш глибокого розуміння ефективності моделей.

Крім того, реалізовано вебзастосунок із клієнт-серверною архітектурою, що складається з клієнтської частини, створеної на основі фреймворку Next.js, та серверної частини, реалізованої із застосуванням Flask, що забезпечує зручний інтерфейс для взаємодії користувача з моделлю генерації SQL-запитів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Уроки від W3Schools українською онлайн. *W3Schools українською. Безплатні уроки онлайн для початківців, школярів та студентів*. URL: https://w3schoolsua.github.io/sql/sql_join.html (дата звернення: 03.06.2025).

2. Gretelai/synthetic_text_to_sql · datasets at hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/datasets/gretelai/synthetic_text_to_sql (дата звернення: 03.06.2025).

3. Singh R. Types of transformer model. *Medium*. URL: https://medium.com/@RobuRishabh/types-of-transformer-model-1b52381fa719 (дата звернення: 03.06.2025).

4. An image is worth 16x16 words: transformers for image recognition at scale / A. Dosovitskiy та ін. *arXiv.org e-Print archive*. URL: https://arxiv.org/pdf/2010.11929 (дата звернення: 03.06.2025).

5. What is a Perceptron? What is the role of bias in a perceptron (or neuron)?. *AIML.com*. URL: https://aiml.com/what-is-a-perceptron/ (дата звернення: 03.06.2025).

6. Ferrer J. How transformers work: a detailed exploration of transformer architecture. *datacamp.com*. URL: https://www.datacamp.com/tutorial/how-transformers-work (дата звернення: 03.06.2025).

7. GeeksforGeeks. BERT Model - NLP - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/ (дата звернення: 03.06.2025).

8. Decoder-Only or encoder-decoder? Interpreting language model as a regularized encoder-decoder / Z. Fu та ін. *arXiv.org*. URL: https://arxiv.org/pdf/2304.04052 (дата звернення: 03.06.2025).

9. Koti V. Evolution of GPT models, GPT 1 to GPT 4. *Medium*. URL: https://medium.com/@vipul.koti333/evolution-of-gpt-models-gpt-1-to-gpt-4-0238ee07a29b (дата звернення: 03.06.2025).

10. D C. R. W. P. Decoder-Only transformers: the workhorse of generative llms. *Deep (Learning) Focus | Cameron R. Wolfe, Ph.D. | Substack*. URL: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse (дата звернення: 03.06.2025).

11. Kuş A., Aci C. Can NLP combined with AI prevent language extinction: a case study of turkish. *ResearchGate*. URL: https://www.researchgate.net/publication/383282857_Can_NLP_Combined_with_AI_Prevent_Language_Extinction_A_Case_Study_of_Turkish (дата звернення: 03.06.2025).

12. MarianMT. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/docs/transformers/en/model_doc/marian (дата звернення: 03.06.2025).

13. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization / J. Zhang та ін. *arXiv.org*. URL: https://arxiv.org/pdf/1912.08777v2 (дата звернення: 03.06.2025).

14. Doshi K. Transformers explained visually (part 2): how it works, step by-step. *towardsdatascience.com*. URL: https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34/ (дата звернення: 03.06.2025).

15. Transformers BART model explained for text summarization. *ProjectPro*. URL: https://www.projectpro.io/article/transformers-bart-model-explained/553 (дата звернення: 03.06.2025).

16. Po L. Encoder-Decoder transformer models: BART and T5. *Medium*. URL: https://medium.com/@lmpo/encoder-decoder-transformer-models-a-comprehensive-study-of-bart-and-t5-132b3f9836ed (дата звернення: 03.06.2025).

17. Doshi K. Audio deep learning made simple (part 1): state-of-the-art techniques. *towardsdatascience.com*. URL: https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504/ (дата звернення: 03.06.2025).

18. Adaloglou N. Why multi-head self attention works: math, intuitions and 10+1 hidden insights | AI Summer. *AI Summer*. URL: https://theaisummer.com/self-attention/ (дата звернення: 03.06.2025).

19. Attention is all you need. *arXiv.org*. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 03.06.2025).

20. Srivastava S. A deep dive into the self-attention mechanism of transformers. *Medium*. URL: https://medium.com/analytics-vidhya/a-deep-dive-into-the-self-attention-mechanism-of-transformers-fe943c77e654 (дата звернення: 03.06.2025).

21. S Y. Deep Learning Series 18: Why we will scale Attention weights. *Medium*. URL: https://medium.com/@yashwanths_29644/deep-learning-series-18-why-we-will-scale-attention-weights-a064f3d8bfcf (дата звернення: 03.06.2025).

22. Introduction - hugging face LLM course. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/learn/llm-course (дата звернення: 03.06.2025).

23. Doshi K. Transformers explained visually (part 3): multi-head attention, deep dive. *Medium*. URL: https://medium.com/data-science/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853 (дата звернення: 03.06.2025).

24. The illustrated transformer. *Jay Alammar – Visualizing machine learning one concept at a time*. URL: https://jalammar.github.io/illustrated-transformer/ (дата звернення: 03.06.2025).

25. Sharma M. What is Add & Norm, as soon as possible?. *Medium*. URL: https://molgorithm.medium.com/what-is-add-norm-as-soon-as-possible-178fc0836381 (дата звернення: 03.06.2025).

26. Hughes C. A brief overview of cross entropy loss. *Medium*. URL: https://medium.com/@chris.p.hughes10/a-brief-overview-of-cross-entropy-loss-523aa56b75d5 (дата звернення: 03.06.2025).

27. In plain english. *plainenglish.io/blog/evaluating-nlp-models-a-comprehensive-guide-to-rouge-bleu-meteor-and-bertscore-metrics-d0f1b1*. URL: https://plainenglish.io/community/evaluating-nlp-models-a-comprehensive-guide-to-rouge-bleu-meteor-and-bertscore-metrics-d0f1b1 (дата звернення: 03.06.2025).

28. deepchecks. Bleu. *www.deepchecks.com*. URL: https://www.deepchecks.com/glossary/bleu/ (дата звернення: 03.06.2025).

29. Chiusano F. Two minutes NLP–Learn the ROUGE metric by examples. *Medium*. URL: https://medium.com/nlplanet/two-minutes-nlp-learn-the-rouge-metric-by-examples-f179cc285499 (дата звернення: 03.06.2025).

30. Banerjee S., Lavie A. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. *CMU School of Computer Science*. URL: https://www.cs.cmu.edu/~alavie/METEOR/pdf/Banerjee-Lavie-2005-METEOR.pdf (дата звернення: 03.06.2025).

31. Transformer architectures - hugging face LLM course. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/learn/llm-course/en/chapter1/6 (дата звернення: 03.06.2025).

32. Humarin/chatgpt_paraphraser_on_T5_base · hugging face. *Hugging Face – The AI community building the future*. URL: https://huggingface.co/humarin/chatgpt_paraphraser_on_T5_base (дата звернення: 03.06.2025).

33. Can chatgpt challenge the scientific impact of published research, particularly in the context of industry 4.0 and smart manufacturing? / V. Terziyan та ін. *Scopus*.

URL: https://www.scopus.com/pages/publications/85189804035 (дата звернення: 10.05.2025).

34. Next.js by vercel - the react framework. *Next.js by Vercel - The React Framework*. URL: https://nextjs.org/ (дата звернення: 03.06.2025).