



LLM-R²: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency

Zhaodonghui Li*

Nanyang Technological University,
DAMO Academy Alibaba Group,
Singapore
G220002@e.ntu.edu.sg

Haitao Yuan†

Nanyang Technological University,
Singapore
haitao.yuan@ntu.edu.sg

Huiming Wang

Singapore University of Technology
and Design, Singapore
huiming_wang@mymail.sutd.edu.sg

Gao Cong

Nanyang Technological University,
Singapore
gaocong@ntu.edu.sg

Lidong Bing

DAMO Academy, Alibaba Group,
Singapore
l.bing@alibaba-inc.com

ABSTRACT

Query rewrite, which aims to improve query efficiency by altering an SQL query's structure without changing its result, has been an important research problem. In order to maintain equivalence between the rewritten query and the original one during rewriting, traditional query rewrite methods always rewrite the queries following certain rewrite rules. However, some problems still remain. First, existing methods of finding the optimal choice or sequence of rewrite rules are still limited and the process always costs a lot of resources. Methods involving discovering new rewrite rules typically require complicated proofs of structural logic or extensive user interactions. Second, current query rewrite methods usually rely highly on DBMS cost estimators which are often not accurate. In this paper, we address these problems by proposing a novel query rewrite method named LLM-R², which leverages a large language model (LLM) to recommend rewrite rules for a database rewrite system. To further enhance the inference ability of the LLM in recommending rewrite rules, we train a contrastive model using a curriculum-based approach to learn query representations and select effective query demonstrations for the LLM. Experimental results show that our method significantly improves the query execution efficiency and outperforms the baseline methods. In addition, our method exhibits high robustness across different datasets.

PVLDB Reference Format:

Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. LLM-R²: A Large Language Model Enhanced Rule-based Rewrite System for Boosting Query Efficiency. PVLDB, 18(1): 53 - 65, 2024.
doi:10.14778/3696435.3696440

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/DAMO-NLP-SG/LLM-R2>.

*Zhaodonghui Li is under the Joint PhD Program between DAMO Academy and Nanyang Technological University

†Haitao Yuan is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vlbd.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:10.14778/3696435.3696440

1 INTRODUCTION

Efficient query processing has been a crucial task in modern database systems. One of the key topics in query optimization is query rewrite [22, 27]. The objective of query rewrite is to generate a new query that is equivalent to the original SQL query but executes in less time. Ideally, query rewrite should meet three critical criteria: (1) **Executability**: the rewritten query should be executable and without any errors; (2) **Equivalence**: it must produce identical results as the original query; (3) **Efficiency**: this encompasses two aspects—*Execution Efficiency* and *Computational Efficiency*. *Execution Efficiency* requires the rewritten query executes more efficiently than the original, while *Computational Efficiency* implies that the overhead of the rewriting process should be justifiable by the time saved during query execution.

To improve both **Executability** and **Equivalence** in rewritten queries, existing studies have mainly focused on rule-based rewriting techniques. In particular, these studies are divided into two complementary research directions: discovering novel rewriting rules and effectively applying existing ones. For the first direction, although additional rewrite rules [7, 34, 36] have been discovered, many challenges remain, particularly concerning the complexity of rule validation and the specificity of their applicability. These challenges often lead to high computational demands and require professional-level user competence. For example, Wetune [34] only supports discovering rewrite rules on limited types of operators and Querybooster [7] necessitates user engagement with specialized rule syntax. This work focuses on the latter direction, exploring methodologies for the effective utilization of pre-established rules. For example, Learned Rewrite [45] utilizes existing rewrite rules from the Apache Calcite [8] platform and learns to select rules to apply. It notably incorporates a Monte Carlo search algorithm together with a machine-learned query cost estimator to streamline the selection process. However, it is non-trivial to solve the challenges related to the computational demand of the Monte Carlo algorithm and the accuracy of the cost estimation model, which can significantly impact the **execution efficiency**.

On the other hand, with the rise of large language models (LLMs), several “large language model for database” projects [3, 37] have emerged, which support direct query rewrite. The idea of these



LLM-R²: 효율성을 극대화하기 위한 대규모 언어 모델 기반 규칙 기반 재작성 시스템

이조동회* 난양 기술
대학교, 다모 아카데미 알리
바바 그룹, 싱가포르 G22000
2@e.ntu.edu.sg

원해태오† 난양 기술
대학, 싱가포르 haitao.yuan
@ntu.edu.sg

왕회명 싱가포르 기술
디자인 대학교, 싱가포르 hui
ming_wang@mymail.sutd.edu
.sg

가오 콩 난양 기술
대학, 싱가포르 gaocong@nt
u.edu.sg

이동병, 다모 아카
데미, 알리바바 그룹, 싱가
포르 l.bing@alibaba-inc.co
m

요약

쿼리 재작성(query rewrite)은 SQL 쿼리의 구조를 변경하여 쿼리 효율성을 개선하면서도 결과를 동일하게 유지하는 것을 목표로 하는 중요한 연구 문제입니다. 재작성 과정에서 원래 쿼리와 동등성을 유지하기 위해, 전통적인 쿼리 재작성 방법은 항상 특정 재작성 규칙에 따라 쿼리를 재작성합니다. 그러나 여전히 몇 가지 문제가 존재합니다. 첫째, 최적의 재작성 규칙 선택이나 순서를 찾는 기준 방법은 제한적이며, 이 과정은 많은 자원을 소모합니다. 새로운 재작성 규칙을 발견하는 방법은 구조적 논리의 복잡한 증명이나 광범위한 사용자 상호작용을 필요로 합니다. 둘째, 현재의 쿼리 재작성 방법은 DBMS 비용 추정기에 크게 의존하는데, 이는 종종 정확하지 않습니다. 이 논문에서는 LLM-R2라는 새로운 쿼리 재작성 방법을 제안하여 이러한 문제를 해결합니다. 이 방법은 대규모 언어 모델(LLM)을 활용하여 데이터베이스 재작성 시스템에 대한 재작성 규칙을 추천합니다. 또한 LLM이 재작성 규칙을 추천하는 주론 능력을 향상시키기 위해, 커리큘럼 기반 접근 방식을 사용하여 대조 모델을 학습시키고 쿼리 표현을 학습하며 LLM을 위한 효과적인 쿼리 시연을 선택합니다. 실험 결과, 이 방법은 쿼리 실행 효율성을 크게 향상시키고 기준 방법보다 우수한 성능을 보입니다. 또한 다양한 데이터셋에서 높은 견고성을 나타냅니다.

PVLDB 참조 형식:

이동희 리, 원해도, 왕회명, 공가오, 병리동. LLM-R²: 쿼리 효율성 향상을 위한 대규모 언어 모델 및 규칙 기반 재작성 시스템. PVLDB, 18(1): 53 - 65, 2024.

doi:10.14778/3696435.3696440

PVLDB 아티팩트 가능성:

소스 코드, 데이터 및/또는 기타 아티팩트는 <https://github.com/DAMO-NLP-SG/LLM-R2>에서 제공됩니다.

*Zhaodonghui Li is under the Joint PhD Program between DAMO Academy and Nanyang Technological University

†Haitao Yuan is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vlbd.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.

doi:10.14778/3696435.3696440

1 소개

효율적인 쿼리 처리는 현대 데이터베이스 시스템에서 중요한 과제입니다. 쿼리 최적화의 핵심 주제 중 하나는 쿼리 다시 쓰기 [22, 27]입니다. 쿼리 다시 쓰기의 목적은 원래 SQL 쿼리와 동등하지만 실행 시간이 더 짧은 새로운 쿼리를 생성하는 것입니다. 이 상적으로, 쿼리 다시 쓰기는 세 가지 중요한 기준을 충족해야 합니다: (1) 실행 가능성: 다시 작성된 쿼리는 실행 가능하고 오류가 없어야 합니다; (2) 동등성: 원래 쿼리와 동일한 결과를 생성해야 합니다; (3) 효율성: 이는 두 가지 측면을 포함합니다. 실행 효율성과 계산 효율성. 실행 효율성은 다시 작성된 쿼리가 원래 쿼리보다 더 효율적으로 실행되어야 함을 의미하는 반면, 계산 효율성은 다시 쓰기 과정의 오버헤드가 쿼리 실행 중 절약된 시간에 의해 정당화되어야 함을 의미합니다.

실행 가능성과 동등성을 모두 향상시키기 위해 재작성된 쿼리에서, 기존 연구들은 주로 규칙 기반 재작성 기술에 초점을 맞춰왔습니다. 특히, 이러한 연구들은 새로운 재작성 규칙을 발견하고 기존 규칙을 효과적으로 적용하는 두 가지 보완적인 연구 방향으로 나뉩니다. 첫 번째 방향에 대해, 추가적인 재작성 규칙들 [7, 34, 36]이 발견되었음에도 불구하고, 규칙 검증의 복잡성과 적용 가능성의 특수성 등과 관련된 많은 도전 과제들이 남아 있습니다. 이러한 도전 과제들은 종종 높은 계산 요구 사항을 초래하고 전문적인 수준의 사용자 역량을 필요로 합니다. 예를 들어, Wetune [34]은 제한된 유형의 연산자에 대해서만 재작성 규칙을 발견하는 것을 지원하고, Querybooster [7]는 사용자가 전문적인 규칙 구문과 상호작용해야 합니다. 이 연구는 후자 방향에 초점을 맞추고, 사전 설정된 규칙의 효과적인 활용을 위한 방법론을 탐구합니다. 예를 들어, Learned Rewrite [45]는 Apache Calcite [8] 플랫폼에서 기존 재작성 규칙을 활용하고, 적용할 규칙을 선택하는 방법을 학습합니다. 이는 몬테카를로 검색 알고리즘과 기계 학습 쿼리 비용 추정기를 함께 통합하여 선택 과정을 간소화합니다. 그러나 몬테카를로 알고리즘의 계산 요구 사항과 비용 추정 모델의 정확성과 관련된 도전 과제를 해결하는 것은 비단 간단한 문제가 아니며, 이는 실행 효율성에 상당한 영향을 미칠 수 있습니다.

다른 한편으로, 대규모 언어 모델(LLMs)의 등장과 함께 데이터베이스를 위한 대규모 언어 모델 여러 프로젝트 [3, 37]가 등장하여 직접적인 쿼리 재작성을 지원합니다. 이러한 프로젝트들의 아이디어는 {v*}

methods is to utilize the sequence-to-sequence generation ability of a language model to directly output a new rewritten query given an input query, without considering any rewrite rules or DBMS information. Although it is possible for these methods to discover new rewrites not following any existing rules, they easily suffer from the hallucination problem of language models [20, 41], especially for long and complicated queries, where language models give plausible but incorrect outputs. Either a syntax or reference error during generation will lead to vital errors when executing the query. Therefore, relying solely on LLM’s output query may violate the **executability** and **equivalence** to the original query, deviating from the basic aim for query rewrite.

To address the limitations of the current query rewriting techniques while benefiting from their advantages, we propose an LLM-enhanced rewrite system. This system uses LLMs to recommend rewrite rules and apply these rules with an existing database platform to rewrite the input query. Inspired by the LLM-based learning framework for using tools [29, 38], we leverage the LLM’s generalization and reasoning abilities for query rewriting while avoiding issues like hallucination. We design a novel LLM-enhanced query rewrite system to automate the process of selecting more effective rewrite. Note that our approach guarantees the **executability** and **equivalence** of the rewritten query since all the candidate rules are provided by existing DB-based rule rewrite platforms. In addition to meeting the basic requirements of valid query rewrite, we also develop new techniques to boost the **execution efficiency** of our rewrite system. Firstly, to overcome hallucination, we collect a pool of demonstrations consisting of effective query rewrites using existing methods and our designed baselines. We develop a contrastive query representation model to select the most useful in-context demonstration for the given query to prompt the system, optimizing the LLM’s selection on rewrite rules. In addition, to address the challenge of limited training data, we propose using the learning curriculum technique [9] to train the model using training data in an easy to hard way. We apply our LLM-enhanced rewrite method on three different datasets, namely TPC-H, IMDB, and DSB. We observe a significant query execution time decrease using our method, requiring only 52.5%, 56.0%, 39.8% of the querying time of the original query and 94.5%, 63.1%, 40.7% of the time of the state-of-the-art baseline method on average on the three datasets. Our main contributions are:

- To the best of our knowledge, this is the first work on an LLM-enhanced query rewrite system that can automatically select effective rules from a given set of rewrite rules to rewrite an input SQL query.
- To enable LLMs to select better rewrite rules for a query, we construct a demonstration pool that contains high-quality demonstrations so that we can select good demonstrations to prompt the LLM-enhanced rewrite system for few-shots learning.
- We develop a contrastive query representation model to optimize the demonstration selection. To address the challenge of limited training data, we further design a learning curriculum to organize the training data from easy to hard.
- We analyze the robustness of our method. By applying our method to unseen datasets and different dataset volumes, we

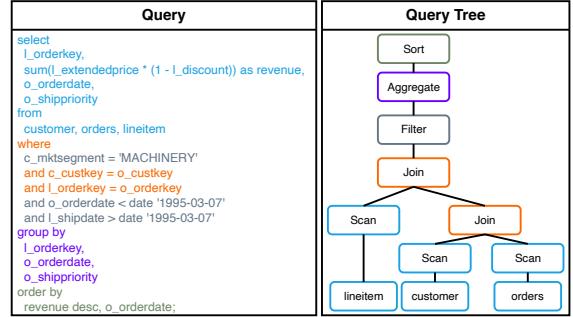


Figure 1: A TPC-H query and its query tree

demonstrate that our method is much more flexible than the baseline methods and shed light on generalizing to other database problems.

2 PRELIMINARY

In this section, we first introduce some key concepts including query, query tree and query rewrite rules in Section 2.1. Then, we will formalize the problem of query rewrite based on rules in Section 2.2. Finally in Section 2.3, we introduce the related work.

2.1 Query and Rewrite Rules

Query & Query tree. Each query in our study is formulated as an executable SQL statement. Furthermore, we model each query as a query tree using various nodes, where each node represents a specific type of query operator (e.g., Sort, Join, and Scan). Figure 1 illustrates an example of a SQL query and its corresponding query tree representation. It is worth noting that any given query can be transformed into a query tree, and conversely, this query tree can be reverted back to its original raw query form.

Query rewrite rules. Given an input query denoted as Q , a sequence of transformation methods, represented as r_1, r_2, \dots , can be applied to the query’s query tree, yielding an equivalent query, denoted as Q^* . These transformation methods, referred to as rewrite rules, encompass a diverse range of functionalities. These include the conversion of one operator to another, the alteration of execution sequences between operators, and the elimination of redundant operators. Table 1 delineates a representative set of these query rewrite rules. For the sake of brevity, we succinctly express the query rewrite process as $Q^* = R(Q)$, where $R = [r_1, r_2, \dots, r_n]$ symbolizes the sequence of n applied rewrite rules.

2.2 Rule-based Query Rewrite

With the introduction of the rewrite rules, we now formally define the problem of query rewrite based on rules as follows:

Definition 2.1. (Rule-based query rewrite): Consider an input query Q and a set of candidate rewrite rules R . The objective is to identify a sequence of rules $R^* = [r_1^*, r_2^*, \dots, r_n^*]$ where $r_i^* \in R$, that transforms the query Q into a more efficient version $Q^* = R^*(Q)$. The efficiency of the rewritten query Q^* is quantified by its execution latency. Such rewrite is characterized by transforming Q into an equivalent query Q^* , which exhibits a lower execution latency compared to other possible rewritten versions of the query.

방법론은 입력 쿼리를 기반으로 새로운 재작성 쿼리를 직접 출력하기 위해 언어 모델의 시퀀스-투-시퀀스 생성 능력을 활용하는 것이다. 이는 기준의 어떤 규칙도 따르지 않는 새로운 재작성 방법을 발견할 수 있지만, 언어 모델의 환각화 문제 [20, 41]를 쉽게 겪는다. 특히 긴 복잡한 쿼리의 경우 언어 모델은 그럴듯하지만 잘못된 출력을 제공한다. 생성 과정에서 구문 또는 참조 오류가 발생하면 쿼리 실행 시 치명적인 오류가 발생한다. 따라서 LLM의 출력 쿼리에만 의존하는 것은 실행 가능성과 원본 쿼리와의 동등성을 훼손하여 쿼리 재작성의 기본 목표에서 벗어날 수 있다.

현재 쿼리 재작성 기술의 한계를 극복하고 그 장점을 활용하기 위해 LLM(대규모 언어 모델) 기반 재작성 시스템을 제안합니다. 이 시스템은 LLM을 사용하여 재작성 규칙을 권장하고, 기존 데이터베이스 플랫폼과 이러한 규칙을 적용하여 입력 쿼리를 재작성합니다. LLM 기반 도구 사용을 위한 학습 프레임워크[29, 38]에서 영감을 받아, 쿼리 재작성 시 LLM의 일반화 및 추론 능력을 활용하면서 환각과 같은 문제를 피합니다. 더 효과적인 재작성을 선택하는 과정을 자동화하기 위해 새로운 LLM 기반 쿼리 재작성 시스템을 설계합니다. 기존 DB 기반 규칙 재작성 플랫폼에서 제공하는 모든 후보 규칙이므로 재작성된 쿼리의 실행 가능성과 동등성을 보장한다는 점에 유의하십시오. 기본적인 유효한 쿼리 재작성 요구 사항을 충족할 뿐만 아니라, 실행 효율성을 높이기 위한 새로운 기술을 개발했습니다. 먼저 환각을 극복하기 위해 기존 방법과 설계된 기준선을 사용하여 효과적인 쿼리 재작성 시연 풀을 수집합니다. 주어진 쿼리를 프롬프트하여 시스템을 안내하는 가장 유용한 인커텍스트 시연을 선택하기 위해 대조적 쿼리 표현 모델을 개발하여 LLM의 재작성 규칙 선택을 최적화합니다. 또한 재한된 학습 데이터의 문제를 해결하기 위해 쉬운 것에서 어려운 것으로 학습 데이터를 학습시키는 학습 커리큘럼 기술[9]을 제안합니다. LLM 기반 재작성 방법을 TPC-H, IMDB, DSB 세 가지 다른 데이터셋에 적용했습니다. 우리 방법은 원래 쿼리 시간의 52.5%, 56.0%, 39.8%와 최신 기준선 방법 시간의 94.5%, 63.1%, 40.7% 만 필요로 하는 상당한 쿼리 실행 시간 감소를 관찰했습니다. 주요 기여는 다음과 같습니다.

- 우리의 최선의 지식에 따르면, 이는 입력 SQL 쿼리를 다시 작성하기 위해 주어진 재작성 규칙 집합에서 효과적인 규칙을 자동으로 선택할 수 있는 LLM 강화 쿼리 재작성 시스템에 대한 최초의 연구입니다.
- LLM이 쿼리에 대한 더 나은 재작성 규칙을 선택할 수 있도록 고품질 데모가 포함된 데모 풀을 구성하여 몇 가지 예시를 통해 LLM 강화 재작성 시스템을 안내할 수 있는 좋은 데모를 선택할 수 있습니다.
- 우리는 시연 선택을 최적화하기 위해 대조적 쿼리 표현 모델을 개발합니다. 제한된 학습 데이터의 문제를 해결하기 위해, 우리는 또한 쉬운 것에서 어려운 것으로 학습 데이터를 구성하기 위한 학습 커리큘럼을 설계합니다.
- 우리는 우리의 방법의 견고함을 분석한다. 미리 보지 못한 데 이터셋과 다른 데이터셋 크기에 우리의 방법을 적용함으로써, 우리는

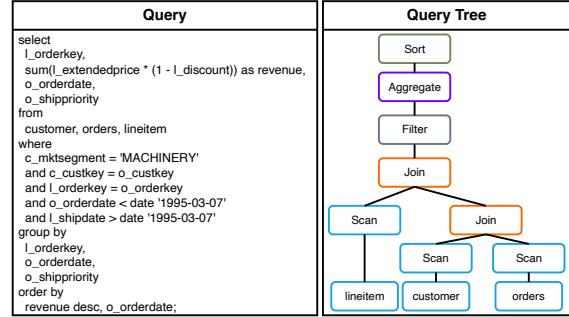


그림 1: TPC-H 쿼리와 그 쿼리 트리

우리의 방법이 기준 방법보다 훨씬 유연하다는 것을 보여주고, 다른 데이터베이스 문제에 일반화하는 데 대한 통찰력을 제공합니다.

2 예비

이 절에서는 섹션 2.1에서 쿼리, 쿼리 트리 및 쿼리 재작성 규칙을 포함한 일부 핵심 개념을 처음 소개합니다. 그런 다음 섹션 2.2에서 규칙 기반 쿼리 재작성 문제를 형식화합니다. 마지막으로 섹션 2.3에서는 관련 작업을 소개합니다.

2.1 쿼리 및 재작성 규칙

쿼리 및 쿼리 트리. 본 연구에서 각 쿼리는 실행 가능한 SQL 문으로 구성됩니다. 나아가, 우리는 다양한 노드를 사용하여 각 쿼리를 쿼리 트리로 모델링하며, 각 노드는 특정 쿼리 연산자 유형 (예: 정렬, 조인 및 스캔)을 나타냅니다. 그림 1은 SQL 쿼리와 그에 상응하는 쿼리 트리 표현의 예를 보여줍니다. 주어진 쿼리는 모두 쿼리 트리로 변환될 수 있으며, 반대로 이 쿼리 트리는 원래의 원시 쿼리 형태로 되돌릴 수 있다는 점에 유의할 가치가 있습니다.

쿼리 재작성 규칙. 입력 쿼리 Q 에 대해, r_1, r_2, \dots 로 표현되는 변환 방법 시퀀스를 쿼리 트리에도 적용하여 동등한 쿼리 Q^* 를 얻을 수 있습니다. 이러한 변환 방법은 재작성 규칙이라고 불리며, 다양한 기능을 포함합니다. 이는 하나의 연산자를 다른 연산자로 변환하거나, 연산자 간의 실행 순서를 변경하거나, 중복된 연산자를 제거하는 것을 포함합니다. 표 1은 이러한 쿼리 재작성 규칙의 대표적인 집합을 나타냅니다. 간결성을 위해 쿼리 재작성 과정을 $Q^* = R(Q)$ 로 간결하게 표현하며, $R = [r_1, r_2, \dots, r_n]$ 는 적용된 재작성 규칙의 시퀀스 n 를 상정합니다.

2.2 규칙 기반 쿼리 다시 쓰기

재작성 규칙의 도입과 함께, 이제 규칙 기반 쿼리 재작성 문제는 다음과 같이 공식적으로 정의합니다:

정의 2.1. (규칙 기반 쿼리 재작성): 입력 쿼리 Q 와 후보 재작성 규칙 집합 R 을 고려한다. 목표는 $R^* = [r_1^*, r_2^*, \dots, r_n^*]$ 와 같은 규칙의 순열을 식별하는 것이다. 여기서 $r_i^* \in R$ 은 쿼리 Q 를 더 효율적인 버전 $Q^* = R^*(Q)$ 으로 변환한다. 재작성된 쿼리 Q^* 의 효율성은 실행 지연 시간으로 양화된다. 이러한 재작성은 Q 를 다른 가능한 재작성된 쿼리 버전보다 실행 지연 시간이 더 짧은 동등한 쿼리 Q^* 로 변환하는 것으로 특징지어진다.

Table 1: Examples of query rewrite rules. Examples of query rewrite rules of the Apache Calcite Rules [1].

Rule Name	Rule Description
AGGREGATE_UNION_AGGREGATE	Rule that matches an Aggregate whose input is a Union one of whose inputs is an Aggregate
FILTER_INTO_JOIN	Rule that tries to push filter expressions into a join condition and into the inputs of the join
JOIN_EXTRACT_FILTER	Rule to convert an inner join to a filter on top of a cartesian inner join
SORT_UNION_TRANSPOSE	Rule that pushes a Sort past a Union

The problem can be formally represented as:

$$\operatorname{argmin}_{R^* \subseteq R} \text{latency}(Q^*) \quad \text{s.t. } Q^* = R^*(Q) \quad (1)$$

2.3 Related Work

2.3.1 Query Rewrite. Query rewrite is a significant function in current Database Management Systems (DBMSs), and can be supported in the query optimizers [16–19, 40]. In particular, DBMSs, such as Calcite [8] and PostgreSQL [4], have developed different rewrite functions to achieve various rewrite rules. Consequently, there are two primary research directions for the query rewriting problem: discovering new rewrite rules and optimally leveraging existing rewrite rules.

Discovering New Rewrite Rules. Recent advancements, exemplified by Querybooster [7] and Wetune [34], have made significant strides in discovering new rewrite rules. Querybooster enables database users to suggest rules through a specialized rule language. On the other hand, Wetune compiles potential rewrite templates and pinpoints constraints that convert these templates into actionable rules. While these methodologies have proven their worth by efficiently handling small real-world workloads, they have their limitations. Querybooster’s effectiveness hinges on the user’s ability to propose potent rules, whereas Wetune’s efficacy on simple or generalized queries remains uncertain.

Selecting Rewrite Rules. The heuristic rewrite approach executes rewrite rules contingent upon the types of operators involved. Learned Rewrite [45] employs a Monte Carlo Tree search to optimize the selection of applicable rules. It conceptualizes each query as a query tree, with applied rules modifying the tree’s structure. This approach utilizes a learned cost model to predict the impact of applying specific rules, enabling the selection of an optimal rewrite sequence through Monte Carlo Tree search. While Learned Rewrite improves adaptability to varying queries and database structures, it faces challenges in cost model accuracy and potential local minima in the search process, highlighting areas for future enhancement in rule-based query rewriting techniques.

2.3.2 LLM-based SQL Solvers. Large Language Models (LLMs) have recently emerged as a hot topic in machine learning research. These models have demonstrated a surprisingly strong ability to handle a variety of text-related tasks such as generation, decision-making, and reasoning. One such task that is highly related to DB research is text-to-SQL, in which an LLM directly generates an SQL query given database information and user requirements. Numerous studies [23, 31, 46] have highlighted the potential of LLMs in the text-to-SQL task, showcasing their proficiency in SQL query-related tasks. While much of this existing research has focused on LLMs’ ability to generate executable queries, there is a growing

recognition of the efficiency and accuracy of these queries. In particular, [23] discussed their attempts in an efficiency-oriented query rewrite task, where an LLM is directly given an input query and tries to rewrite it into a more efficient one. However, a significant issue previous methods face is the problem of hallucination, which refers to instances where the model generates incorrect outputs but is done so with a misleading level of confidence. Although some methods try to utilize instruction tuning to solve the problem, there are still three main limitations. First, only open-source LLMs like Llama and Phi can be fine-tuned, but they lag behind closed-source LLMs like the GPT family. Second, fine-tuning techniques still cannot eliminate hallucinations. One example is the SOTA LLM for txt-to-SQL task Granite-20b-code model [26], which is an LLM specially fine-tuned on txt-to-SQL data but only 67.86% of the queries generated by the model on the BIRD benchmark [23] are executable. This is particularly problematic in the context of database applications, where accuracy is paramount and it motivates us to use rule-based methods to ensure query correctness. Lastly, leveraging LLMs’ generalization ability is crucial. Instruction tuning requires re-tuning with new datasets and queries, which is inefficient. Therefore, we propose a different direction of utilising the LLMs while overcoming hallucination by using LLM’s in-context learning capability and adopt a DB-based SQL rewriter enhanced by an LLM.

2.3.3 In-context Learning. Due to the extensive data and resource requirements of fine-tuning an LLM, many works choose to utilize LLMs by in-context learning (ICL). The concept of ICL, first introduced by Brown et al. in their seminal work on GPT-3 [10], shows that language models like GPT-3 can leverage in-context demonstrations at inference time to perform specific tasks, without updating the model weights. ICL typically involves enriching the context with select examples to steer the model’s output. Formally, consider a model denoted as M and a contextual input represented by P . The output o generated by applying the ICL method to model M with input P can be succinctly expressed as $o = ICL_M(P)$.

ICL has rapidly gained popularity for addressing diverse challenges in natural language processing. However, it is a sophisticated technique requiring careful implementation. Extensive research, including studies by [35] and [24], has explored the intricacies of LLMs’ learning processes in this context. These studies highlight that the success of in-context learning is closely related to the construction of the context and the quality of the examples used.

2.3.4 Contrastive Learning by Curriculum. Contrastive learning by curriculum merges the strengths of contrastive learning and curriculum learning to create efficient machine learning models with minimal labeled data. Contrastive learning enhances representation by bringing similar data points closer and separating dissimilar ones, while curriculum learning structures the training process progressively. In particular, the SOTA method [39] in natural language processing creates data with different levels of difficulties using the PCA jittering method to form the learning curriculum. However, PCA jittering is a method that generates textual sentences and cannot be applied to generating SQL queries. Similarly, in computer vision, the SOTA methods [11] and [33] demonstrate how a curriculum can be set up to incrementally learn a classification model. They use the curriculum to incrementally train models by starting

표 1: 쿼리 재작성 규칙의 예. Apache Calcite 규칙 [1]의 쿼리 재작성 규칙의 예.

Rule Name	Rule Description
AGGREGATE_UNION_AGGREGATE	Rule that matches an Aggregate whose input is a Union one of whose inputs is an Aggregate
FILTER_INTO_JOIN	Rule that tries to push filter expressions into a join condition and into the inputs of the join
JOIN_EXTRACT_FILTER	Rule to convert an inner join to a filter on top of a cartesian inner join
SORT_UNION_TRANSPOSE	Rule that pushes a Sort past a Union

문제는 다음과 같이 형식적으로 표현할 수 있습니다:

$$\operatorname{argmin}_{R^* \subseteq R} \text{latency}(Q^*) \quad \text{s.t. } Q^* = R^*(Q) \quad (1)$$

2.3 관련 연구

2.3.1 허용 다시 작성하기. 쿼리 다시 작성 기능은 현재 데이터베이스 관리 시스템(DBMS)에서 중요한 기능이며, 쿼리 최적화기[16-19, 40]에서 지원할 수 있습니다. 특히 Calcite[8] 및 PostgreSQL[4]와 같은 DBMS는 다양한 다시 작성 규칙을 달성하기 위해 여러 다시 작성 기능을 개발했습니다. 따라서 쿼리 다시 작성 문제에 대한 두 가지 주요 연구 방향은 새로운 다시 작성 규칙을 발견하고 기존 다시 작성 규칙을 최적으로 활용하는 것입니다.

새로운 재작성 규칙 발견하기. Querybooster[7]와 Wetune[34]와 같은 최근의 발전은 새로운 재작성 규칙을 발견하는 데 상당한 진전을 이루었습니다. Querybooster는 데이터베이스 사용자가 전문화된 규칙 언어를 통해 규칙을 제안할 수 있게 합니다. 반면, Wetune는 잠재적인 재작성 템플릿을 컴파일하고 이러한 템플릿을 실행 가능한 규칙으로 변환하는 제약 조건을 파악합니다. 이러한 방법론은 작은 실제 작업 부하를 효율적으로 처리함으로써 그 가치를 입증했지만, 한계가 있습니다. Querybooster의 효과성은 사용자가 강력한 규칙을 제안할 수 있는 능력에 달려 있는 반면, Wetune의 유효성은 간단하거나 일반화된 쿼리에 대해 불확실합니다.

재작성 규칙 선택. 경험적 재작성 접근 방식은 관련된 연산자의 유형에 따라 재작성 규칙을 실행합니다. 학습된 재작성[45]은 적용 가능한 규칙의 선택을 최적화하기 위해 몬테카를로 트리 검색을 사용합니다. 이는 각 쿼리를 쿼리 트리로 개념화하고, 적용된 규칙이 트리의 구조를 수정합니다. 이 접근 방식은 학습된 비용 모델을 사용하여 특정 규칙을 적용하는 영향을 예측하여, 몬테카를로 트리 검색을 통해 최적의 재작성 순서를 선택할 수 있게 합니다. 학습된 재작성은 다양한 쿼리와 데이터베이스 구조에 대한 적응성을 향상시키지만, 비용 모델의 정확도와 검색 과정에서의 국소 최적화 문제에 직면하여, 규칙 기반 쿼리 재작성 기술의 향후 개선 영역을 강조합니다.

2.3.2 LLM 기반 SQL 솔버. 최근 대형 언어 모델(LLM)이 기계 학습 연구에서 뜨거운 주제로 부상했습니다. 이러한 모델은 생성, 의사 결정, 추론과 같은 다양한 텍스트 관련 작업에서 놀라울 정도로 강력한 능력을 보여주었습니다. DB 연구와 밀접하게 관련된 작업 중 하나는 텍스트-투-SQL로, LLM이 데이터베이스 정보와 사용자 요구 사항을 기반으로 직접 SQL 쿼리를 생성합니다. 여러 연구 [23, 31, 46]는 LLM이 텍스트-투-SQL 작업에서 잠재력을 보여주며 SQL 쿼리 관련 작업에서 뛰어난 능력을 입증했습니다. 기존 연구의 대부분은 LLM이 실행 가능한 쿼리를 생성하는 능력에 초점을 맞추고 있지만, 이 분야에 대한 관심은 점점 커지고 있습니다.

이러한 쿼리의 효율성과 정확성 인식. 특히, [23]은 효율성 지향 쿼리 재작성 작업에서 그들의 시도를 논의했는데, 여기서 LLM은 입력 쿼리를 직접 받아 더 효율적인 것으로 재작성하려고 시도합니다. 그러나 이전 방법들이 직면한 중요한 문제는 환각 문제입니다. 이는 모델이 잘못된 출력을 생성하지만, 이를 매우 확신에 찬 수준으로 수행하는 경우를 말합니다. 일부 방법은 지침 튜닝을 활용하여 이 문제를 해결하려고 시도하지만, 여전히 세 가지 주요 한계가 있습니다. 첫째, Llama와 Phi와 같은 오픈 소스 LLM만 미세 조정할 수 있지만, 이들은 GPT 가족과 같은 비공개 LLM에 비해 뒤처집니다. 둘째, 미세 조정 기술은 여전히 환각을 제거할 수 없습니다. 한 예는 txt-to-SQL 작업용 SOTA LLM인 Granite-20b-code 모델[26]으로, 이는 txt-to-SQL 데이터에 특별히 미세 조정된 LLM이지만, BIRD 벤치마크[23]에서 모델이 생성한 쿼리의 67.86%만 실행 가능합니다. 이는 데이터베이스 애플리케이션의 맥락에서 특히 문제가 되는데, 정확성이 최우선이며, 이는 쿼리 정확성을 보장하기 위해 규칙 기반 방법을 사용하도록 동기를 부여합니다. 마지막으로, LLM의 일반화 능력을 활용하는 것이 중요합니다. 지침 튜닝은 새로운 데이터셋과 쿼리로 재튜닝이 필요하므로 비효율적입니다. 따라서 우리는 LLM의 컨텍스트 내 학습 능력을 활용하여 환각을 극복하면서 LLM을 활용하는 다른 방향을 제안하고, LLM으로 강화된 DB 기반 SQL 재작성 기능을 채택합니다.

2.3.3 컨텍스트 내 학습. LLM을 "미세 조정하는 데 필요한 광범위한 데이터와 자원 때문에, 많은 연구들은 컨텍스트 내 학습(ICL)을 통해 LLM을 활용하는 것을 선택합니다. ICL의 개념은 Brown 외 연구자들이 GPT-3에 대한 선구적인 연구[10]에서 처음 소개하였습니다. 이는 GPT-3와 같은 언어 모델이 주로 시 컨텍스트 내 시연을 활용하여 특정 작업을 수행할 수 있음을 보였습니다. 이는 모델 가중치를 업데이트하지 않고도 가능합니다. ICL은 일반적으로 모델의 출력을 유도하기 위해 컨텍스트를 선택된 예시로 풍부하게 만드는 것을 포함합니다. 형식적으로, M 로 표시된 모델과 P 로 표현된 컨텍스트 입력을 고려해 봅시다. ICL 방법은 모델 M 에 입력 P 에 적용하여 생성된 출력 o 는 간결하게 $o = ICL_M(P)$ 로 표현될 수 있습니다.

ICL은 자연어 처리의 다양한 과제를 해결하는 데 있어 빠르게 인기를 얻고 있습니다. 그러나 이는 세심한 구현이 필요한 정교한 기술입니다. [35] 및 [24]의 연구들을 포함한 광범위한 연구는 이 맥락에서 LLM의 학습 과정의 복잡성을 탐구해 왔습니다. 이러한 연구들은 컨텍스트 학습의 성공이 컨텍스트 구성과 사용된 예제의 품질과 밀접한 관련이 있음을 강조합니다.

2.3.4 교과 과정에 의한 대조 학습. 교과 과정에 의한 대조 학습은 대조 학습과 교과 과정 학습의 장점을 결합하여 최소한의 레이블이 지정된 데이터로 효율적인 기계 학습 모델을 생성합니다. 대조 학습은 유사한 데이터 포인트를 가까이 가져오고 다른 포인트를 분리하여 표현을 향상시키며, 교과 과정 학습은 훈련 과정을 섬진적으로 구조화합니다. 특히, 자연어 처리에서 SOTA 방법[39]은 PCA 혼들림 방법을 사용하여 다양한 난이도 수준의 데이터를 생성하여 학습 교과 과정을 형성합니다. 그러나 PCA 혼들림은 텍스트 문장을 생성하는 방법이며 SQL 쿼리 생성에 적용할 수 없습니다. 마찬가지로 컴퓨터 비전에서는 SOTA 방법[11] 및 [33]이 분류 모델을 점진적으로 학습하기 위해 교과 과정을 어떻게 설정하는지 보여줍니다. 그들은 교과 과정을 사용하여 모델을 점진적으로 훈련하기 위해 시작합니다.

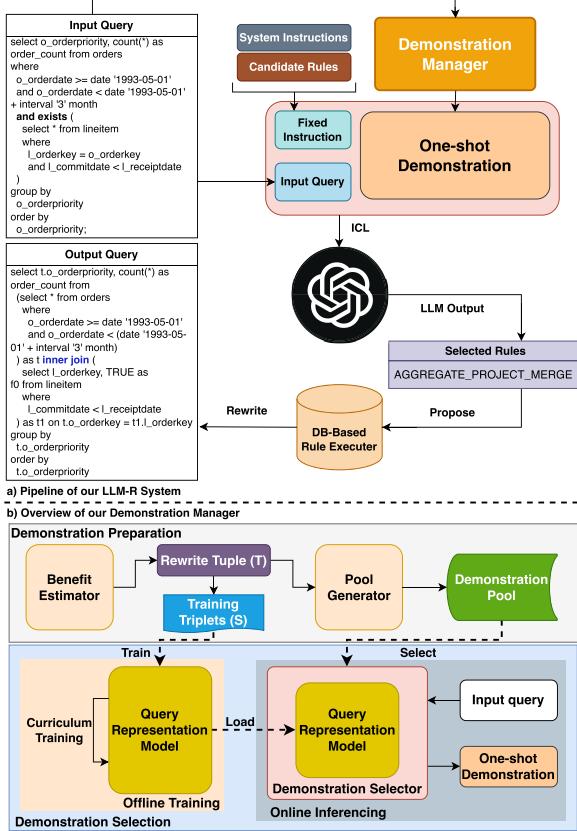


Figure 2: The Framework of LLM-enhanced Rewrite System

to focus on more confidently labeled data. However, they assume a semi-supervised setting where a computer vision model generating image pseudo-labels is required. Therefore, the SOTA methods may not directly apply to our problem and we need to adapt the idea of contrastive learning by curriculum to our own problem.

3 LLM-ENHANCED REWRITE SYSTEM

In this section, we will introduce our innovative LLM-enhanced rule-based rewrite system (**LLM-R²**). In Section 3.1, we will first illustrate the pipeline of our rewrite system. Then in Section 3.2, we will state our motivation to optimize the demonstration selection and introduce our novel *Demonstration Manager* module.

3.1 System Pipeline

As shown in Figure 2(a), the system integrates an LLM into the query rewrite system utilizing the ICL methodology [10]. We construct the ICL prompt with three main components:

Input query: We employ the SQL statement corresponding to the provided input query Q for the prompt construction.

Fixed instruction: The fixed instruction consists of a system instruction I and a rule instruction R . While the system instruction specifies the task requirements, the rule instruction includes a comprehensive list of all candidate rewrite rules available for the language model to select. Each rule is accompanied by a concise explanation, enabling informed decision-making.

One-shot demonstration: Similar to directly letting LLMs rewrite queries, selecting rewrite rules using LLMs may also easily suffer from the hallucination problem, like outputting non-existing rules. To mitigate this and ensure the LLMs' outputs are more closely aligned with our task requirements, yielding superior rule suggestions, we use the demonstration as a part of the prompt. Formally, we define our demonstration given to the LLM-R² system as a pair of text $D = \langle Q^D, R^D \rangle$, where Q^D is the example query assembling the input query and $R^D = [r_1^D, \dots]$ is the list of rules that have been applied to rewrite the example query. Such demonstrations can successfully instruct the LLM to follow the example and output a list of rewrite rules to apply on the new input query. In particular, this involves selecting a high-quality demonstration D from many successful rewritten demonstrations (i.e., denoted as a pool \mathcal{D}) for each input query to guide the LLM effectively. To achieve this goal, we design a module named *Demonstration Manager*, whose details are elucidated in the subsequent section.

As specifically highlighted, Figure 3 delineates the prompt utilized within the In-Context Learning (ICL) process of our system. Upon constructing the prompt and feeding it into the LLM, we can extract a sequence of rewrite rules from the model's output. These rules undergo further processing and execution by a database-based rule executor. For instance, the original input query in Figure 2(a) is modified by the “AGGREGATE_PROJECT_MERGE” rule, as highlighted in bold. This modification transforms the original query into a more optimized output query, demonstrating the practical application and effectiveness of the extracted rules in query optimization processes. Through the synergy of the LLM's superior generalization capabilities and the rule executor's precision, our proposed system guarantees extensive applicability, alongside ensuring the executability and equivalence of the rewritten queries. Consequently, this rewrite process can be formalized as follows:

Definition 3.1. (LLM-enhanced Query Rewrite): Given a large language model M , a textual instruction outlining the rewrite task I , a set of candidate rules R , one successful rewrite demonstration D selected from the demonstration pool \mathcal{D} , and an input query Q , a prompt P is constructed and provided as input to M as $P = I \oplus R \oplus D \oplus Q$. From M , a sequence of rewrite rules R^* is derived:

$$R^* = ICL_M(P)$$

By sequentially applying these rewrite rules R^* , we generate an optimally equivalent query, represented as $Q^* = R^*(Q)$.

3.2 Demonstration Manager Overview

Motivation. In the above ICL process, optimizing the prompt $P = I \oplus R \oplus D \oplus Q$ is crucial for improving the output quality of LLMs. Given the fixed settings of system instruction(I), rule instruction(R), and input query(Q), our optimization efforts focus primarily on the demonstration(D), which is chosen to enhance model performance. Recent studies on LLMs (e.g., [10, 35]) have underscored the positive impact of high-quality in-context demonstrations on LLM output, reducing the tendency of LLMs to produce hallucinatory content. As shown in Figure 4, our rewrite system exhibits similar effectiveness variability w.r.t. the demonstrations used, further emphasizing the necessity of optimizing demonstration selection for specific input queries. Therefore, it is an important problem to optimize

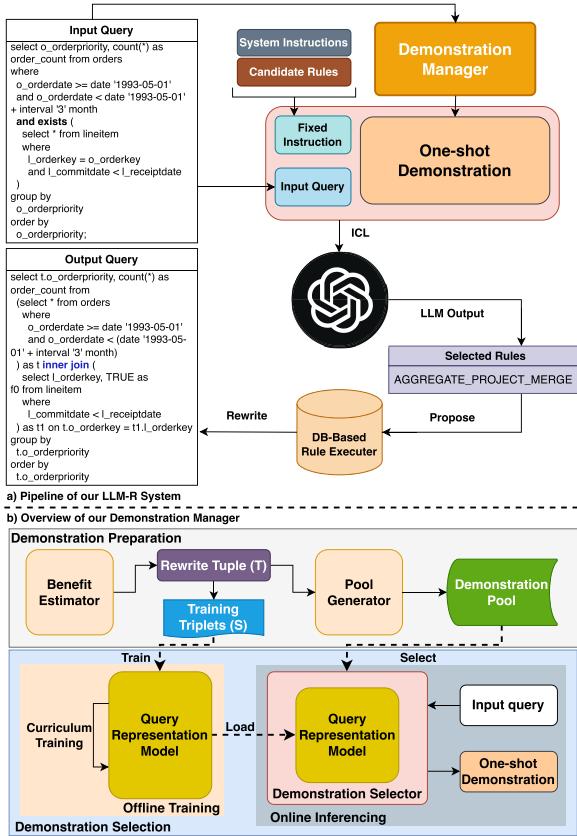


그림 2: LLM 기반 재작성 시스템의 프레임워크

더 신뢰할 수 있는 레이블이 지정된 데이터에 집중하기 위해. 그러나 그들은 컴퓨터 비전 모델이 이미지 가짜 레이블을 생성하는 준감독 설정을 가정합니다. 따라서 최첨단(SOTA) 방법은 우리의 문제에 직접 적용될 수 없으며, 우리는 대조 학습의 아이디어를 우리의 문제에 맞게 조정해야 합니다.

3 LLM-향상된 다시 쓰기 시스템

이 절에서는 혁신적인 LLM 기반 규칙 기반 재작성 시스템(LLM-R2)을 소개합니다. 3.1절에서는 재작성 시스템의 파이프라인을 먼저 설명합니다. 그 다음으로 3.2절에서는 시연 선택 최적화에 대한 동기를 설명하고 새로운 시연 관리자 모듈을 소개합니다.

3.1 시스템 파이프라인

그림 2(a)에서 볼 수 있듯이, 이 시스템은 ICL 방법론 [10]을 활용하여 LLM을 쿼리 재작성 시스템에 통합합니다. 우리는 세 가지 주요 구성 요소를 사용하여 ICL 프롬프트를 구성합니다:

입력 쿼리: 제공된 입력 쿼리 Q 에 대응하는 SQL 문을 프롬프트 생성에 사용합니다.

고정된 명령어: "xed 명령어는 시스템 명령어 I 와 규칙 명령어 R 로 구성됩니다. 시스템 명령어는 작업 요구 사항을 지정하고, 규칙 명령어는 언어 모델이 선택할 수 있는 모든 후보 재작성 규칙의 포괄적인 목록을 포함합니다. 각 규칙에는 간결한 설명이 함께 제공되어 정보에 입각한 의사 결정을 가능하게 합니다.

단번에 시연: LLMs가 쿼리를 직접 다시 작성하도록 허용하는 것과 유사하게, LLMs를 사용하여 다시 작성 규칙을 선택하는 것도 비현실적인 규칙을 출력하는 것과 같은 환경 문제를 쉽게 겪을 수 있습니다. 이를 완화하고 LLMs의 출력이 작업 요구 사항과 더 밀접하게 일치하도록 하여 우수한 규칙 제안을 유도하기 위해, 우리는 시연을 프롬프트의 일부로 사용합니다. 형식적으로, 우리는 LLM-R2 시스템에 제공하는 시연을 입력 쿼리와 Q^D 를 구성하는 예제 쿼리의 쌍 $D = \langle Q^D, R^D \rangle$ 으로 정의합니다. 여기서 $R^D = [r^D_1, \dots]$ 은 예제 쿼리를 다시 작성하기 위해 적용된 규칙 목록입니다. 이러한 시연은 LLM이 예를 따라 새로운 입력 쿼리에 적용할 다시 작성 규칙 목록을 출력하도록 성공적으로 지시 할 수 있습니다. 특히, 이는 많은 성공적으로 다시 작성된 시연(즉, 풀 \mathcal{D} 로 표시)에서 고품질 시연 D 를 선택하는 것을 포함하며, 이를 통해 각 입력 쿼리에 대해 LLM을 효과적으로 안내합니다. 이 목표를 달성하기 위해, 우리는 후속 섹션에서 세부 사항이 설명된 Demonstration Manager라는 모듈을 설계했습니다.

특히 강조된 것처럼, 그림 3은 우리 시스템의 인컨텍스트 러닝 (ICL) 과정에서 사용된 프롬프트를 나타냅니다. 프롬프트를 구성하고 이를 LLM에 입력하면, 모델의 출력에서 일련의 재작성 규칙을 추출할 수 있습니다. 이러한 규칙은 데이터베이스 기반 규칙 실행자에 의해 추가 처리 및 실행됩니다. 예를 들어, 그림 2(a)의 원본 입력 쿼리는 "AGGREGATE_PROJECT_MERGE" 규칙에 의해 수정되며, 이는 굽게 강조되어 있습니다. 이 수정은 원본 쿼리를 더 최적화된 출력 쿼리로 변환하여 추출된 규칙의 실용적인 적용과 쿼리 최적화 과정에서의 효과를 보여줍니다. LLM의 뛰어난 일반화 능력과 규칙 실행자의 정밀함의 시너지를 통해, 우리의 제안된 시스템은 광범위한 적용 가능성을 보장할 뿐만 아니라 재작성된 쿼리의 실행 가능성과 동등성을 확보합니다. 따라서 이 재작성 과정은 다음과 같이 형식화할 수 있습니다:

정의 3.1. (LLM-강화 쿼리 재작성): 주어진 큰 언어 모델 M , 재작성 작업 개요를 설명하는 텍스트 지시사항 I , 후보 규칙 집합 R , 시연 풀 \mathcal{D} 에서 선택된 성공적인 재작성 시연 D , 그리고 입력 쿼리 Q , 프롬프트 P 가 구성되고 M 에 입력으로 제공됩니다. $P = I \oplus R \oplus D \oplus Q$ 로. M 로부터 재작성 규칙의 순서 R^* 가 도출됩니다.

$$R^* = ICL_M(P)$$

이 재작성 규칙 R^* 을 순차적으로 적용하여 최적으로 동등한 쿼리를 생성하고, 이는 $Q^* = R^*(Q)$ 으로 표현됩니다.

3.2 데모스트레이션 매니저 개요

동기. 위에서 설명한 ICL 프로세스에서 시스템 지시어(I)와 규칙 지시어(R) 및 입력 쿼리(Q)의 "고정된 설정"을 고려할 때, LLM의 출력 품질을 향상시키는 데 있어 프롬프트($P = I \oplus R \oplus D \oplus Q$) 최적화가 매우 중요합니다. 최근 LLM에 대한 연구(예: [10, 35])는 고품질 인컨텍스트 데모스트레이션이 LLM 출력에 미치는 긍정적인 영향을 강조했으며, LLM이 환각성 콘텐츠를 생성하는 경향을 줄였습니다. 그림 4에서 볼 수 있듯이, 우리의 리라이팅 시스템은 사용된 데모스트레이션에 따라 유사한 효과 변동성을 보이며, 이는 특정 입력 쿼리에 대한 데모스트레이션 선택 최적화의 필요성을 더욱 강조합니다. 따라서 이를 최적화하는 것은 중요한 문제입니다.

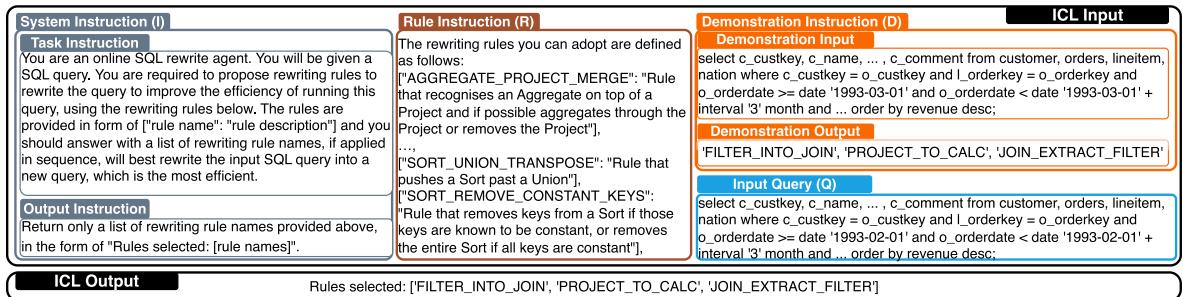


Figure 3: An Example of the In-Context Learning Process in LLM-R². All the instructions are concatenated together as one string input to the LLM. In a zero-shot setting, the “Demonstration Instruction” will be removed and an input query will be appended directly after the “Rule Instruction”.



Figure 4: Example of good and bad demonstration selections

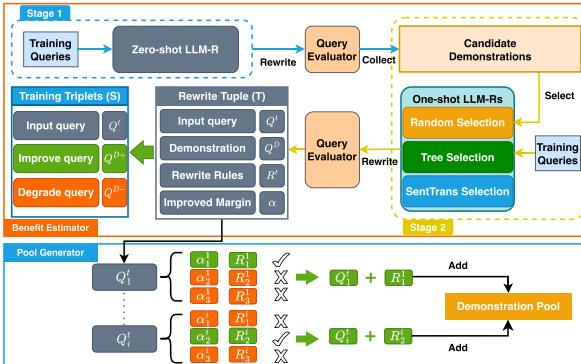


Figure 5: Our demonstration preparation module generates a set of training triplets and a demonstration pool.

the demonstration selected for a given input query. Particularly, we address this problem by designing the *Demonstration Manager* module.

Overview. Figure 2(b) illustrates the basic structure of our proposed *Demonstration Manager* module, comprising two parts: *Demonstration Preparation* and *Demonstration Selection*.

(1) The primary objective of the *Demonstration Preparation* is to generate a substantial number of successful rewritten demonstrations for constructing a demonstration pool. Furthermore, this part also serves to supply training data essential for model learning in the second part. Specifically, we design two modules: the *Benefit Estimator* and the *Pool Generator*, to achieve our objectives. The *Benefit Estimator* is capable of assessing the potential benefits of a given query rewrite strategy, thereby generating corresponding rewrite tuple recording the performance of this rewrite strategy on the input query. Subsequently, the *Pool Generator* is employed to extract demonstrations for constructing a pool. Moreover, we utilize the rewrite tuples to derive training triplets, which are essential for model learning in subsequent parts.

(2) The second part involves the *Demonstration Selection* module, tasked with identifying the optimal demonstration from the pool

for each input query. This process is enhanced by incorporating a query representation model within the selector, designed to evaluate the similarity between input queries and demonstrations in the pool. This representation model undergoes offline training using the training data. In addition, to obtain an effective model, we enhance the model’s training through the integration of a curriculum learning approach. Afterwards, the trained model is integrated into *Demonstration Selector* for online inference. In other words, upon receiving an input query for rewriting, the selector discerns and selects the most appropriate demonstration from the pool based on the trained model. More detailed elaboration on the above two parts will be provided in the following sections.

4 DEMONSTRATION PREPARATION

In this section, we aim to generate sufficient high-quality data to build the demonstration pool. As shown in Figure 5, we first design the *Benefit Estimator* module to generate the ground truth, where each ground truth data point indicates the efficiency gain obtained by rewriting an input query using generated rules in the context of a demonstration. With sufficient ground truth, including both good and bad samples, we further design the *Pool Generator* module to select all good samples to build the demonstration pool. In addition, we can deduce contrastive training triplets from the ground truth, which can help train our selection model.

4.1 Benefit Estimator

Since we are only able to start with solely training queries without demonstrations, the triplet generation pipeline is segmented into two distinct phases: the first stage involves initializing high quality candidate demonstrations utilizing baseline method and a zero-shot LLM-R² system where no demonstration is selected, followed by the demonstration adoption stage employing a one-shot LLM-R² system. Subsequently, each stage is elucidated in detail.

Stage-1: We start with a diverse set of input queries collected from our dataset as the training set. To obtain a rich set of effective rewrites as candidate demonstrations, we first apply our zero-shot LLM-enhanced rewrite system (LLM-R²) to rewrite the training set queries. After getting the rewrite rules adopted and the resulted rewrite queries, we directly execute the rewritten queries on the corresponding databases. The execution time of the rewritten queries as well as the original queries is evaluated to collect the initial

System Instruction (I)	Rule Instruction (R)	Demonstration Instruction (D)	ICL Input
<p>Task Instruction: You are an online SQL rewrite agent. You will be given a SQL query. You are required to propose rewriting rules to rewrite the query to improve the efficiency of running this query, using the rewriting rules below. The rules are provided in form of ["rule name": "rule description"] and you should answer with a list of rewriting rule names, if applied in sequence, will best rewrite the input SQL query into a new query, which is the most efficient.</p> <p>Output Instruction: Return only a list of rewriting rule names provided above, in the form of "Rules selected: [rule names]".</p>	<p>The rewriting rules you can adopt are defined as follows:</p> <ul style="list-style-type: none"> ["AGGREGATE_PROJECT_MERGE": "Rule that recognises an Aggregate on top of a Project and if possible aggregates through the Project or removes the Project"], ... ["SORT_UNION_TRANSPOSE": "Rule that pushes a Sort past a Union"], ... ["SORT_REMOVE_CONSTANT_KEYS": "Rule that removes keys from a Sort if those keys are known to be constant, or removes the entire Sort if all keys are constant"]. 	<p>Demonstration Input select c_custkey, c_name, ... , c_comment from customer, orders, lineitem, nation where c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate >= date '1993-03-01' and o_orderdate < date '1993-03-01' + interval '3' month and ... order by revenue desc;</p> <p>Demonstration Output 'FILTER_INTO_JOIN', 'PROJECT_TO_CALC', 'JOIN_EXTRACT_FILTER'</p>	<p>Input Query (Q) select c_custkey, c_name, ... , c_comment from customer, orders, lineitem, nation where c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate >= date '1993-02-01' and o_orderdate < date '1993-02-01' + interval '3' month and ... order by revenue desc;</p>

그림 3: LLM-R2의 컨텍스트 내 학습 과정의 예시. 모든 지침은 하나의 문자열 입력으로 LLM에 연결됩니다. 제로샷 설정에서는 "데모 지침"이 제거되고 "규칙 지침" 바로 뒤에 입력 쿼리가 직접 추가됩니다.

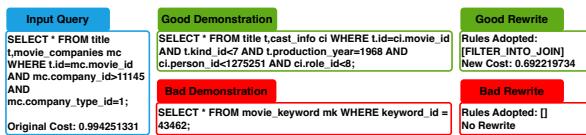


그림 4: 좋은 시연 선택과 나쁜 시연 선택의 예시

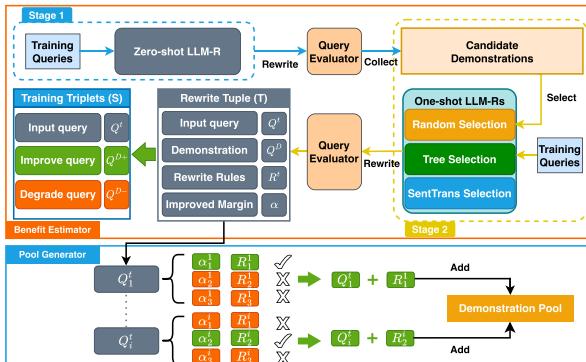


그림 5: 우리의 시연 준비 모듈은 훈련 트리플 집합과 시연 풀을 생성합니다.

특히, 우리는 시연 관리자 모듈을 설계함으로써 이 문제를 해결합니다. 주어진 입력 쿼리에 대한 시연을 선택합니다.

개요. 그림 2(b)는 우리가 제안하는 데몬스트레이션 관리자 모듈의 기본 구조를 보여주며, 이는 데몬스트레이션 준비와 데몬스터레이션 선택의 두 부분으로 구성됩니다.

(1) 시연 준비의 주요 목표는 시연 풀을 구성하기 위해 많은 성공적인 다시 작성된 시연을 생성하는 것입니다. 또한 이 부분은 모델 학습에 필수적인 훈련 데이터를 공급하는 역할도 합니다. 구체적으로, 우리는 목표 달성을 위해 두 가지 모듈인 혜택 추정기(Benefit Estimator)와 풀 생성기(Pool Generator)를 설계했습니다. 혜택 추정기는 주어진 쿼리 다시 작성 전략의 잠재적 이점을 평가할 수 있으며, 이를 통해 입력 쿼리에 대한 이 다시 작성 전략의 성능을 기록하는 대응하는 다시 작성 튜플을 생성합니다. 그 후, 풀 생성기가 풀을 구성하기 위해 시연을 추출하는 데 사용됩니다. 또한, 우리는 다시 작성 튜플을 사용하여 후속 부분의 모델 학습에 필수적인 훈련 트리플을 유도합니다.

(2) 두 번째 부분은 시연 선택 모듈과 관련이 있으며, 이는 풀에서 최적의 시연을 식별하는 것을 담당합니다.

각 입력 쿼리마다 이 과정은 쿼리 표현 모델을 선택기에 통합하여 입력 쿼리와 풀의 데모 간의 유사성을 평가하도록 설계함으로써 향상됩니다. 이 표현 모델은 훈련 데이터를 사용하여 오프라인으로 훈련됩니다. 또한 효과적인 모델을 얻기 위해 교과 학습 접근 방식을 통합하여 모델의 훈련을 강화합니다. 그 후, 훈련된 모델은 온라인 추론을 위해 데모 선택기에 통합됩니다. 다시 말해, 다시 쓰기를 위한 입력 쿼리를 수신하면 선택기는 훈련된 모델을 기반으로 풀에서 가장 적절한 데모를 식별하고 선택합니다. 위 두 부분에 대한 자세한 설명은 다음 섹션에서 제공됩니다.

4 시연 준비

이 섹션에서는 데모 풀을 구축하기 위한 충분한 고품질 데이터를 생성하는 것을 목표로 합니다. 그림 5와 같이, 먼저 이득 추정기 모듈을 설계하여 지상 진실을 생성합니다. 각 지상 진실 데이터 포인트는 데모의 맵락에서 생성된 규칙을 사용하여 입력 쿼리를 다시 작성함으로써 얻는 효율성 향상을 나타냅니다. 충분한 지상 진실, 즉 좋은 샘플과 나쁜 샘플 모두를 확보한 후, 풀 생성기 모듈을 설계하여 모든 좋은 샘플을 선택하여 데모 풀을 구축합니다. 또한, 지상 진실에서 대조적 학습 트리플을 유추할 수 있으며, 이는 우리의 선택 모델을 훈련하는 데 도움이 됩니다.

4.1 이점 추정기

우리는 데모 없이 훈련 쿼리만으로 시작할 수 있기 때문에, 삼중 항 생성 파이프라인은 두 가지 명확한 단계로 분할됩니다: 첫 번째 단계는 기본 방법과 제로샷 LLM-R2 시스템을 사용하여 고품질 후보 데모를 초기화하는 것이고, 데모가 선택되지 않습니다. 그 다음은 데모 채택 단계를 거치며, 원샷 LLM-R2 시스템을 사용합니다. 이후 각 단계는 자세히 설명됩니다.

1단계: 우리는 데이터셋에서 수집한 다양한 입력 쿼리 집합으로 시작하여 훈련 세트로 사용합니다. 효과적인 재작성 후보 시연을 얻기 위해, 먼저 제로샷 LLM 강화 재작성 시스템(LLM-R2)을 적용하여 훈련 세트 쿼리를 재작성합니다. 재작성 규칙을 적용하고 결과 재작성 쿼리를 얻은 후, 직접 재작성된 쿼리를 해당 데이터베이스에서 실행합니다. 재작성된 쿼리와 원본 쿼리의 실행 시간을 평가하여 초기 데이터를 수집합니다.

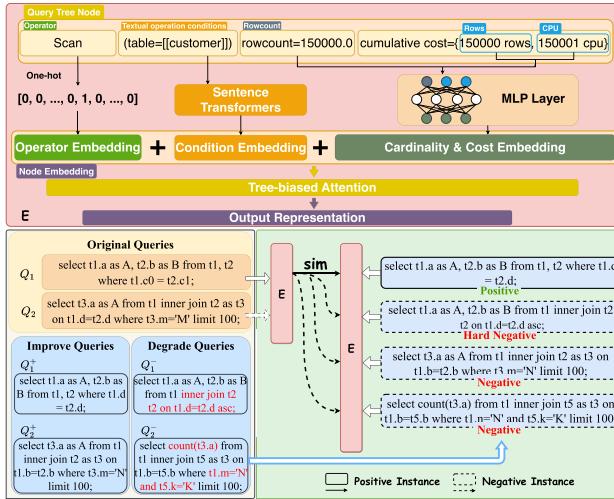


Figure 6: Our representation model encodes each query tree node into a fixed-length vector, with the final query representation obtained through tree-biased attention over the nodes. The model E is trained using contrastive query tuples.

candidate demonstration set consisting of the improvable queries, together with their rules adopted.

Stage-2: With the candidate demonstrations collected from the previous step, we can then estimate the benefits of these demonstrations when they are selected for a given input query. Motivated by [35], such improvable demonstrations are supposed to be more useful for the LLM to output improving rewrite suggestions, compared to using any degraded rewrite queries as demonstrations. In addition, the more “similar” the improving demonstration query is to the input query, the better output the LLM will generate. However, different from natural language inputs’ simple textual similarity, the similarity between SQL queries is indeed more complicated. To identify if the pool we collected truly contains high-quality and “similar” demonstrations for new input queries and refine the demonstration pool, we designed three heuristic demonstration-selection methods based on different levels of similarity as follows.

- **Random Selection:** A random demonstration query is selected from the candidate demonstrations for a given input query, where the similarity level lies on the same input category.
- **Tree Selection:** Query tree is an important structural feature for the queries, therefore, it is natural to align similarity with the query tree structure. We first compute the query trees of all the candidate demonstration queries, with operators as the tree nodes. Given an input query, we select the demonstration with the minimum tree edit distance from the input query tree within the candidate demonstrations.
- **SentTrans Selection:** At the textual level, we observe that queries are always considered as sentences for the language models to process. Based on the observation, we treat input queries as sentences and select the candidate demonstration query whose embedding is the most similar to the input query. Most of the effective LLMs are closed-sourced, which means we are not able to obtain the query embeddings of such LLMs. However, similar to LLMs, some small pre-trained language models

share the same sequence-to-sequence mechanism, that the input text is first encoded by an encoder before feeding to the model. Using such encoders, like Sentence Transformers [28], we can obtain an embedding of a given sentence.

With the three demonstration selection methods above, we can prompt our **LLM-R²** system with the one-shot demonstration to obtain various rewrite results on the same training set. These new rewrite queries from the one-shot **LLM-R²** system are then evaluated in the same way as in Stage-1. Specifically, when we adopt one-shot demonstration to rewrite an input query Q^t , we are able to estimate the benefit obtained from the demonstration by constructing the rewrite tuples (T) as (Q^t, D, R^t, α) , where Q^t represents a training query, D is the demonstration (Q^D, R^D) selected for Q^t , R^t denotes the adopted rules for Q^t , and α represents the improved margin obtained by the query rewrite. In particular, given the original query cost C_0 and the cost of rewritten query C_r , we define the improved margin as $\alpha = C_0/C_r$, where the larger margin the better rewrite result and larger benefit we have.

In addition, a set of training triplets is generated using the rewrite tuples obtained in preparation for training a contrastive representation model. For a given query Q^t in the rewrite tuple (Q^t, D, R^t, α) , we consider the demonstration query Q^D adopted as an improve query Q^{D+} for Q , if the improved margin $\alpha > 1$. In contrast, we denote the demonstration query as a degrade query Q^{D-} if $\alpha < 1$. If there are multiple improve(degrade) queries, we only select the one with the largest(smallest) improved margin. Since we have adopted multiple one-shot selection methods, now we are able to construct a training triplet for a given query as (Q^t, Q^{D+}, Q^{D-}) . A set of training triplets can be further constructed if we enumerate the whole training query set.

4.2 Pool Generator

Apart from the training triplets, we also hope to prepare an effective demonstration pool so that our learned demonstration selection model can select demonstrations from it during online inference. The rewrite tuple generated by the *Benefit Estimator* module, recording the effectiveness of a sequence of rewrite rules R^t on an input query Q^t , naturally fits our need for a high-quality rewrite demonstration.

In particular, given the set of rewrite tuples generated by n input queries, we first separate them into n groups $\{T_i\}_{1 \leq i \leq n}$ based on their corresponding input queries. Therefore, each group T_i can be represented as the tuple set $\{(Q_i^t, D_1^i, R_1^i, \alpha_1^i), (Q_i^t, D_2^i, R_2^i, \alpha_2^i), \dots\}$. Since we have adopted various methods, multiple tuples have the same input query, and we only need the optimal rewrite rule sequence to form a demonstration for the query. Therefore, for each training query Q_i^t and its corresponding tuple group T_i , we only select the tuple with the largest improved margin, and the order is denoted as $*$, which can be formulated as follows:

$$* = \operatorname{argmax}_{j \in [1, |T_i|]} \alpha_j^i \quad (2)$$

$$\text{s.t. } T_i = \{(Q_i^t, D_1^i, R_1^i, \alpha_1^i), (Q_i^t, D_2^i, R_2^i, \alpha_2^i), \dots\}$$

Next, we construct the demonstration containing the input query and rules as the pair (Q_i^t, R_*^i) , and then add the demonstration to the pool. As shown in Figure 5, when the largest improved margins α_1^i and α_2^i are identified for input queries Q_1^t and Q_2^t , the corresponding

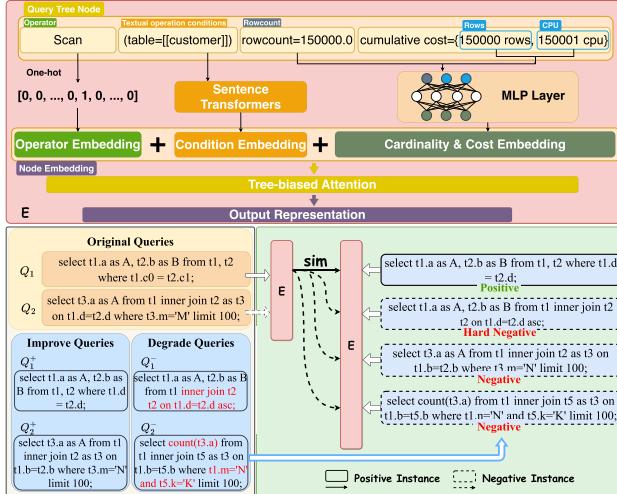


그림 6: 우리의 표현 모델은 각 쿼리 트리 노드를 고정 길이의 벡터로 인코딩하며, 최종 쿼리 표현은 노드에 대한 트리 편향 주의(tree-biased attention)를 통해 얻습니다. 이 모델 E 는 대조적 쿼리 튜플을 사용하여 학습됩니다.

개선 가능한 쿼리들로 구성된 후보 시연 집합과 채택된 규칙들.

단계-2: 이전 단계에서 수집한 후보 시연들을 사용하여 주어진 입력 쿼리에서 이러한 시연들의 이점을 추정할 수 있습니다. [35]에 의해 동기를 부여받은 이러한 개선 가능한 시연들은 저하된 리라이트 쿼리를 시연으로 사용하는 것보다 LLM이 개선된 리라이트 제안을 출력하는 데 더 유용하다고 가정됩니다. 또한, 개선된 시연 쿼리가 입력 쿼리와 더 "유사" 할수록 LLM은 더 나은 출력을 생성할 것입니다. 그러나 자연어 입력의 단순한 텍스트 유사성과 달리, SQL 쿼리 간의 유사성은 실제로 더 복잡합니다. 수집한 풀이 새로운 입력 쿼리에 대해 고품질이고 "유사"한 시연을 실제로 포함하는지 확인하고 시연 풀을 정제하기 위해, 우리는 다음과 같은 다양한 유사성 수준에 기반한 세 가지 경험적 시연 선택 방법을 설계했습니다.

- **임의 선택:** 주어진 입력 쿼리에 대한 후보 시연 중에서 입력 카테고리가 동일한 유사성 수준에 있는 임의 시연 쿼리가 선택됩니다.
- **나무 선택:** 쿼리 나무는 쿼리에 대한 중요한 구조적 특징이므로, 쿼리 나무 구조와의 유사성을 맞추는 것이 자연스럽습니다. 먼저 모든 후보 시연 쿼리에 대한 쿼리 나무를 계산하고, 연산자를 나무의 노드로 사용합니다. 입력 쿼리가 주어지면, 입력 쿼리 나무와 후보 시연 중 최소 트리 편집 거리를 가진 시연을 선택합니다.
- **텍스트 수준에서,** 우리는 쿼리가 항상 언어 모델이 처리해야 할 문장으로 간주된다는 것을 관찰합니다. 이 관찰에 기반하여, 우리는 입력 쿼리를 문장으로 처리하고 입력 쿼리와 가장 유사한 임베딩을 가진 후보 데모스트레이션 쿼리를 선택합니다. 효과적인 대형 언어 모델(LLM)의 대부분은 소스 코드 공개가 되지 않아 이러한 LLM의 쿼리 임베딩을 얻을 수 없습니다. 그러나 LLM과 유사하게, 일부 작은 사전 학습된 언어 모델은 ...

같은 순차-순차 메커니즘을 공유하며, 입력 텍스트는 모델에 입력하기 전에 인코더에 의해 "rst 인코딩"됩니다. 이러한 인코더, 예를 들어 문장 트랜스포머 [28]를 사용하면 주어진 문장의 임베딩을 얻을 수 있습니다.

위의 세 가지 시연 선택 방법 중 하나를 사용하여 LLM-R2 시스템에 원샷 시연을 제공하여 동일한 훈련 세트에 대한 다양한 다시 쓰기 결과를 얻을 수 있습니다. 이러한 새로운 다시 쓰기 쿼리는 단계 1과 동일한 방식으로 평가됩니다. 특히, 입력 쿼리 Q^t 를 다시 쓰기 위해 원샷 시연을 채택할 때, 시연을 통해 얻은 이점을 추정할 수 있습니다. 재구성된 다시 쓰기 튜플 (T)은 (Q^t, D, R^t, α) 로 구성함으로써, 여기서 Q^t 는 훈련 쿼리를 나타내고, D 은 Q^t 에 대해 선택된 시연 (Q^D, R^D) 을 나타내며, R^t 은 Q^t 에 대해 채택된 규칙을 나타내고, α 는 쿼리 다시 쓰기를 통해 얻은 향상된 여백을 나타냅니다. 특히, 원래 쿼리 비용 C_0 과 다시 쓰기된 쿼리의 비용 C_t 가 주어지면, 향상된 여백을 $\alpha = C_0/C_t$ 로 정의합니다. 여기서 더 큰 여백은 더 나은 다시 쓰기 결과와 더 큰 이점을 의미합니다.

추가로, 재작성 튜플을 얻어 대조적 표현 모델을 훈련시키기 위해 훈련 삼중항 집합이 생성됩니다. 주어진 재작성 튜플 (Q^t, D, R^t, α) 내의 쿼리 Q^t 에 대해, 개선된 여백 $\alpha >$ 가 1보다 크면, 우리는 개선 쿼리 Q^{D+} 로 채택된 데모 쿼리 Q^D 를 고려합니다. 반대로, $\alpha <$ 가 1보다 작으면, 우리는 그것을 열화 쿼리 Q^{D-} 로 표시합니다. 개선(열화) 쿼리가 여러 개 있다면, 우리는 가장 큰(가장 작은) 개선 여백을 가진 것만 선택합니다. 여러 개의 일회용 선택 방법을 채택했으므로, 이제 주어진 쿼리에 대해 (Q^t, Q^{D+}, Q^{D-}) 와 같은 훈련 삼중항을 구성할 수 있습니다. 전체 훈련 쿼리 집합을 열거하면, 더 많은 훈련 삼중항 집합을 구성할 수 있습니다.

4.2 풀 제너레이터

훈련 튜플 외에 온라인 추론 중 학습된 데몬스트레이션 선택 모델이 선택할 수 있는 효과적인 데몬스트레이션 풀을 준비하기를 희망합니다. 베이트 추정기 모듈이 생성한 리라이트 튜플은 입력 쿼리 Q^t 에 대한 일련의 리라이트 규칙 R^i 의 효율성을 기록하여 고품질 리라이트 데몬스트레이션에 대한 요구 사항에 자연스럽게 부합합니다.

특히, n 입력 쿼리에 의해 생성된 리라이팅 튜플 집합을, 해당 입력 쿼리에 따라 n 그룹 $\{T_i\}_{1 \leq i \leq n}$ 으로 분리합니다. 따라서 각 그룹 T_i 는 튜플 집합 $\{(Q_i^t, D_i^t, R_i^t, \alpha_i^t), (Q_i^t, D_i^t, R_i^t, \alpha_i^t), \dots\}$ 을 표현할 수 있습니다. 다양한 방법을 채택했기 때문에 여러 튜플이 동일한 입력 쿼리를 가지며, 쿼리에 대한 시연을 구성하기 위해 필요한 것은 최적의 리라이팅 규칙 시퀀스입니다. 따라서 각 학습 쿼리 Q^t 와 그에 해당하는 튜플 그룹 T_i 에 대해, 가장 크게 향상된 마진을 가진 튜플만 선택하고, 그 순서는 *로 표기됩니다. 이는 다음과 같이 공식화할 수 있습니다:

$$* = \operatorname{argmax}_{j \in [1, |T_i|]} \alpha_j^i \quad (2)$$

$$\text{s.t. } T_i = \{(Q_i^t, D_i^t, R_i^t, \alpha_i^1), (Q_i^t, D_i^t, R_i^t, \alpha_i^2), \dots\}$$

다음으로, 입력 쿼리와 규칙을 쌍 (Q^t, R^i) 로 포함하는 시연을 구성하고, 그 후 시연을 풀에 추가합니다. 그림 5와 같이, 입력 쿼리 Q^t 과 Q^t 에 대해 가장 큰 개선 여유 α^{t1} 과 α^{t2} 가 식별되면, 이에 대응하는

demonstrations $\langle Q_1^t, R_1^1 \rangle$ and $\langle Q_1^t, R_2^1 \rangle$ are selected with the rewrite rules R_1^1 and R_2^1 adopted.

5 DEMONSTRATION SELECTION

Motivation. Addressing the challenge of enhancing system performance, the selection of an optimal rewrite demonstration to guide the LLM for any given input query is required and remains uncertain. Intuitively, the greater the “similarity” between the input and demonstration queries, the more applicable the rewrite rule, thereby enhancing the LLM’s output efficacy. Therefore, to capture such “similarity”, we design a contrastive model to learn the representations of queries in this *Demonstration Selection* module, where better demonstration queries are to have more similar representations to the input query. Consequently, the demonstration query that exhibits the highest resemblance to the input query is selected for the LLM, optimizing the generation of more effective outputs.

Overview. In order to learn a contrastive representation model efficiently and effectively, the selection module consists of two main components: our contrastive model and a curriculum learning pipeline to improve the model training. We will first outline the representation model and its contrastive learning structure in Section 5.1, followed by a detailed discussion of the whole model learning pipeline in Section 5.2.

5.1 Contrastive Representation Model

As shown in Figure 6, our representation model E is constructed as a query encoder to encode the information describing a query, and a contrastive training structure to further train the encoder given training data. In particular, the information of a query tree is first encoded by nodes into node embeddings. A tree-biased attention layer will then compute the final representation of the query given the node embeddings. Such an encoder E is then trained using the contrastive learning structure drawn below it.

Query encoder. The representation of a query should focus on various key attributes, like the query tree structure and columns selected. Therefore, we design an encoder following [42] to take the query trees generated by DBMS’ query analyzer as inputs. It is notable that the original encoding in [42] utilizes the physical query plan which contains richer information, so that the objective of estimating query cost can be successfully achieved. Since we aim to capture the similarity between queries, we refer to [44] and separately encode the following information for each query tree node instead in our encoder, as shown in the top half of Figure 6:

- **Operator type:** We use one-hot encoding to encode the operator types into one vector, with value one for the current node operator type and zero for the rest positions.
- **Operator conditions:** Within each node, the details for the operator are explained in parentheses, including sort order for “Sort” operator, selected column for “Scan” operator etc. Different from the physical plans used in [42], such information has no unified form for encoding. We consider the conditions as text and encode using a pre-trained Sentence Transformers encoder [28]. Such an encoder can capture the textual differences between conditions effectively and have unified embedding dimensions to simplify further analysis.

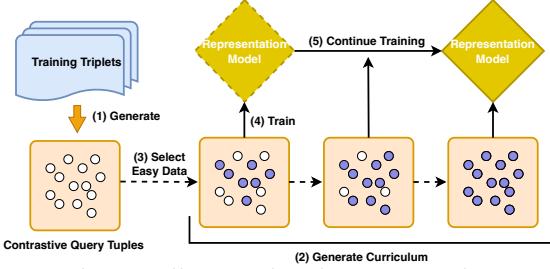


Figure 7: The overall curriculum learning pipeline to train the contrastive selector using generated training triplets.

- **Cardinality and cost:** From [43] we observe that the estimated cardinality and cost are important in describing a query. We collect the row count and estimated cumulative cost values and normalise them through an MLP layer.

We simply concatenate the three information vectors together to be the encoded embedding for a node in the given query tree. We use the same tree Transformer model in [42] to get the final representation of a query given its tree nodes’ embeddings. The final representation of the whole query will be computed by the tree-biased attention module.

Contrastive learning structure. Due to the necessity of executing queries, the volume of training triplets produced by our demonstration preparation module is limited. Unlike the query representation model in [42], which is trained directly on abundant labeled data, our approach requires a more sophisticated training framework to effectively capture query representation with the generated training data. Inspired by SimCSE [15], we design a contrastive learning structure to train our query representation model on the limited training data. In a training batch containing N tuples, we consider each original query’s improved query as its “positive” query, its degraded query as its “hard negative” query, and the remaining improved and degraded queries within the same batch as “negative” queries. This allows us to pull close distances between original queries and their improved versions while pushing apart those with degraded queries. Following such setting, the loss l_i for the i^{th} tuple (Q_i, Q_i^+, Q_i^-) can be computed as

$$l_i = -\log \frac{e^{\text{sim}(h_i, h_i^+)/\tau}}{\sum_{j=1}^N (e^{\text{sim}(h_i, h_j^+)/\tau} + e^{\text{sim}(h_i, h_j^-)/\tau})} \quad (3)$$

where τ is a temperature hyper-parameter, h_i , h_i^+ and h_i^- stand for the representation of Q_i , Q_i^+ and Q_i^- respectively, and the function $\text{sim}(h_1, h_2)$ is the cosine similarity $\frac{h_1^T h_2}{\|h_1\| \cdot \|h_2\|}$.

As an example in a training batch of size 2, for the first original query Q_1 shown in the bottom part of Figure 6, the positive query will be its corresponding improve query Q_1^+ , and other in-batch improve or degrade queries Q_1^-, Q_2^+ and Q_2^- are all regarded as negative queries. The final loss for the batch will be the sum of the losses for the two tuples.

5.2 Curriculum Learning Pipeline

Motivation. Although we have developed a representation-based demonstration selector, training the contrastive model presents several challenges. First, unlike the original SimCSE approach used in natural language inference tasks, which benefits from abundant

시위 $\langle Q_1^t, R_{11} \rangle$ 및 $\langle Q_i^t, R_2^i \rangle$ 는 재작성 규칙 R_{11} 및 R_2^i 를 사용하여 선택됩니다.

5 데몬스트레이션 선택

동기. 시스템 성능을 향상시키는 과제를 해결하기 위해, 주어진 입력 쿼리에 대한 LLM을 안내하기 위한 최적의 다시 쓰기 시연을 선택하는 것이 필요하며 여전히 불확실합니다. 직관적으로, 입력 및 시연 쿼리 간의 "유사성"이 클수록 다시 쓰기 규칙이 더 적용 가능하며, 이는 LLM의 출력 효율을 향상시킵니다. 따라서 이러한 "유사성"을 포착하기 위해, 우리는 이 시연 선택 모듈에서 쿼리의 표현을 학습하기 위해 대조적 모델을 설계합니다. 여기서 더 나은 시연 쿼리는 입력 쿼리와 더 유사한 표현을 가져야 합니다. 그 결과, 입력 쿼리와 가장 닮은 시연 쿼리가 LLM에 선택되어 더 효과적인 출력 생성을 최적화합니다. 개요. 대조적 표현 모델을 효율적이고 효과적으로 학습하기 위해, 선택 모듈은 두 가지 주요 구성 요소로 구성됩니다: 우리의 대조적 모델과 모델 학습을 개선하기 위한 교육 과정 학습 파이프라인입니다. 먼저 5.1절에서 표현 모델과 그 대조적 학습 구조를 개요하고, 이어서 5.2절에서 전체 모델 학습 파이프라인에 대해 상세히 논의합니다.

5.1 대조적 표현 모델

그림 6과 같이, 우리의 표현 모델 E 은 쿼리 인코더로 구성되며, 이는 쿼리를 설명하는 정보를 인코딩하고, 대조적 훈련 구조를 통해 훈련 데이터를 기반으로 인코더를 추가로 훈련합니다. 특히, 쿼리 트리의 정보는 노드에 의해 "rst 인코딩되어 노드 임베딩이 생성됩니다. 그 후, 트리 편향 주의(attention) 계층이 노드 임베딩을 기반으로 쿼리의 "최종 표현"을 계산합니다. 이러한 인코더 E 는 그 아래에 그려진 대조적 학습 구조(contrastive learning structure)를 사용하여 훈련됩니다.

쿼리 인코더. 쿼리의 표현은 쿼리 트리 구조와 선택된 열과 같은 다양한 핵심 속성에 초점을 맞추어야 합니다. 따라서, 우리는 DBMS의 쿼리 분석기에 의해 생성된 쿼리 트리를 입력으로 받아들이는 인코더를 [42]를 따라 설계했습니다. 주목할 만한 점은 [42]의 원래 인코딩이 더 풍부한 정보를 포함하는 물리적 쿼리 계획을 활용하여 쿼리 비용을 추정하는 목표를 성공적으로 달성한다는 것입니다. 우리는 쿼리 간의 유사성을 포착하기를 원하므로, [44]를 참조하여 그림 6의 상단 절반에 표시된 대로 각 쿼리 트리 노드에 대해 다음과 같은 정보를 별도로 인코딩합니다:

- 연산자 유형: 현재 노드 연산자 유형에는 1을, 나머지 위치에는 0을 할당하는 원-핫 인코딩(one-hot encoding)을 사용하여 연산자 유형을 하나의 벡터로 인코딩합니다.
- 연산자 조건: 각 노드 내에서 연산자의 세부 사항은 괄호 안에 설명되어 있으며, "정렬" 연산자의 정렬 순서, "스캔" 연산자의 선택된 열 등 포함됩니다. [42]에서 사용되는 물리적 계획과 달리, 이러한 정보는 인코딩을 위한 통일된 형식이 없습니다. 우리는 조건을 텍스트로 간주하고 사전 학습된 문장 변환기 인코더 [28]를 사용하여 인코딩합니다. 이러한 인코더는 조건 간의 텍스트 차이를 효과적으로 포착하고 추가 분석을 간소화하기 위해 통일된 임베딩 차원을 가집니다.

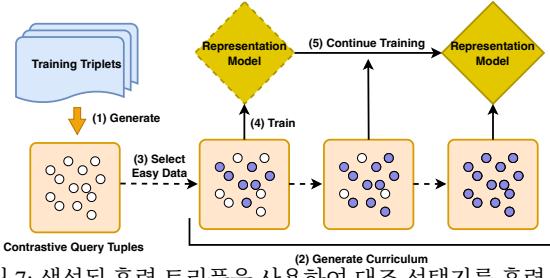


그림 7: 생성된 훈련 트리플을 사용하여 대조 선택기를 훈련시키기 위한 전체 커리큘럼 학습 파이프라인.

- 카디널리티와 비용: [43]에서 관찰하듯이, 추정 카디널리티와 비용은 쿼리를 설명하는 데 중요합니다. 우리는 행 수와 추정 누적 비용 값을 수집하고 MLP 계층을 통해 정규화합니다.

우리는 주어진 쿼리 트리에서 노드에 대한 인코딩된 임베딩이 되도록 세 개의 정보 벡터를 단순히 연결합니다. 우리는 [42]에서 동일한 트리 트랜스포머 모델을 사용하여 트리 노드 임베딩을 고려하여 주어진 쿼리의 "최종 표현"을 얻습니다. 전체 쿼리의 "최종 표현"은 트리 기반 주의 모듈에 의해 계산될 것입니다.

대조 학습 구조. 쿼리 실행의 필요성으로 인해, 우리 데모 준비 모듈에 의해 생성된 훈련 트리플렛의 양은 제한적입니다. [42]에 있는 쿼리 표현 모델처럼 풍부한 레이블이 지정된 데이터에 직접 훈련되는 것과 달리, 우리의 접근 방식은 생성된 훈련 데이터와 함께 쿼리 표현을 효과적으로 포착하기 위해 더 정교한 훈련 프레임워크를 필요로 합니다. SimCSE [15]에 영감을 받아, 우리는 제한된 훈련 데이터에서 우리의 쿼리 표현 모델을 훈련시키기 위해 대조 학습 구조를 설계합니다. N 튜플을 포함하는 훈련 배치에서, 우리는 각 원본 쿼리의 개선된 쿼리를 그 "긍정" 쿼리로, 저하된 쿼리를 그 "어려운 음수" 쿼리로 간주하며, 같은 배치 내의 나머지 개선되고 저하된 쿼리들을 "음수" 쿼리로 간주합니다. 이를 통해 원본 쿼리와 그 개선된 버전 사이의 거리를 좁히고, 저하된 쿼리들과의 거리를 벌릴 수 있습니다. 이러한 설정에 따라, i 손실은 i_{th} 튜플 (Q_i, Q_i^+, Q_i^-) 에 대해 계산될 수 있습니다.

$$l_i = -\log \frac{e^{\text{sim}(h_i, h_i^+)/\tau}}{\sum_{j=1}^N (e^{\text{sim}(h_i, h_j^+)/\tau} + e^{\text{sim}(h_i, h_j^-)/\tau})} \quad (3)$$

여기서 τ 은 온도 하이퍼파라미터이고, h_i, h_i^+ 및 h_i^- 은 각각 Q_i, Q_i^+ 및 Q_i^- 의 표현을 나타내며, $\text{sim}(h_1, h_2)$ 함수는 코사인 유사도 $\|h_1\| \|h_2\|$ 입니다.

예를 들어, 크기 2인 훈련 배치에서, 그림 6의 하단에 표시된 "원래 쿼리 Q_1 "를 위해, 양수 쿼리는 그에 해당하는 개선 쿼리 Q_1^+ 가 될 것이고, 배치 내 다른 개선 또는 악화 쿼리 Q_1^-, Q_2^+ , 그리고 Q_2^- 는 모두 음수 쿼리로 간주됩니다. 배치에 대한 최종 손실은 두 튜플의 손실 합계입니다.

5.2 커리큘럼 학습 파이프라인

동기. 우리는 표현 기반 데몬스트레이션 선택기를 개발했지만, 대조적 모델을 훈련시키는 데는 여러 가지 어려움이 있다. 첫째, 자연어 추론 작업에서 사용되는 원래 SimCSE 접근 방식과 달리, 풍부한 데이터를 활용할 수 없다.

data [12], our model’s training is constrained by data scarcity. Our contrastive query tuples, derived from a limited variety of training triplets, face scalability issues due to the high computational cost of query execution. Furthermore, the complexity of query representations in our model surpasses the simplicity of word embeddings used in SimCSE. Given these constraints—limited data and a complex training target—we propose adopting a curriculum learning pipeline. This approach is designed to enhance the learning efficiency and effectiveness of our contrastive representation model.

Algorithm 1 Contrastive Training under Curriculum Scheduler

Require: Total training data S_0 , Number of iterations I
Require: Initialized model E_0

```

1:  $N_0 = \text{len}(S_0)$                                 ▷ Total number of training data
2:  $N = \lceil (N_0/I) \rceil$                           ▷ Each iteration incremental data size
3:  $Tr_0 = \emptyset$                                  ▷ Initial training data
4:  $i = 1$                                          ▷ Initial iteration
5: while  $i \leq I$  do
6:   if  $\text{len}(N) > \text{len}(S_{i-1})$  then           ▷ If less than N data left
7:      $Tr_i \leftarrow S_{i-1}$                          ▷ Select all the data left
8:      $Tr_i = Tr_{i-1} + Tr_i$                       ▷ Append to training data
9:     Train  $E_{i-1}$  on  $Tr_i$  and get  $E_i$           ▷ Continue training
10:    else
11:       $C_i \leftarrow Top_N(S_{i-1})$  based on  $\text{conf}_{E_{i-1}}(\cdot)$  ▷ Select N easy data
         following curriculum from the unvisited dataset
12:       $S_i = S_{i-1} - C_i$                          ▷ Deduct them from unvisited data
13:       $Tr_i = Tr_{i-1} + C_i$                       ▷ Append them to training data
14:      Train  $E_{i-1}$  on  $Tr_i$  and get  $E_i$           ▷ Train the model
15:       $i = i + 1$                                ▷ Move to next iteration
16:    end if
17: end while
18: Use the final  $E_I$  for inference

```

As depicted in Figure 7, the essence of this pipeline is to strategically implement an effective curriculum. Starting with the provided training triplets, we initially train our contrastive representation model on a smaller, simpler subset, progressively incorporating easier subsets from the remaining dataset and retraining the model until all training data is utilized. The methodology for generating our curriculum is detailed in Algorithm 1. This algorithm begins with an empty model; each iteration involves selecting a subset of training data on which the current model performs with the highest confidence, followed by model retraining to incorporate this new subset (lines 5–17). This iterative retraining process continues until the entire training dataset has been incorporated.

In particular, we sample the easier subset of remaining training data by the confidence of the model to the data. Suppose we get the embeddings of two queries using our contrastive model to be x and y , we can compute their similarity scores using the cosine similarity to keep consistency with the training objective in Equation 3. For each contrastive query tuple $\langle Q, Q^+, Q^- \rangle$, since we expect to have the $\text{sim}(E(Q), E(Q^+)) = 1$ and $\text{sim}(E(Q), E(Q^-)) = 0$, we define a confidence score of the contrastive model E to a given tuple as:

$$\text{conf}_E(Q) = \text{sim}(E(Q), E(Q^+)) - \text{sim}(E(Q), E(Q^-)) + 1 \quad (4)$$

Therefore, at each iteration i , given our trained model E_{i-1} , previous training dataset T_{i-1} and the unvisited dataset D_{i-1} , we can generate the current tuples (denoted as S_i) with the highest confidence score in D_{i-1} . They are then moved into the training set,

resulting in the new training set $T_i = T_{i-1} + S_i$ and the new unvisited dataset $D_i = D_{i-1} - S_i$.

6 EXPERIMENT

In this section, we evaluate our proposed system’s effectiveness, efficiency, and generalization capabilities.

6.1 Experimental Setup

6.1.1 Dataset. We use three datasets from different domains for our evaluations:

IMDB (JOB workload) [21]: The IMDB [25] dataset consists of data on movies, TV shows, and actors. It’s utilized in conjunction with the Join Order Benchmark (JOB) to test a database management system’s efficiency in executing complex join queries, and it comprises 5,000 queries.

TPC-H [5]: A benchmark dataset for evaluating database management systems, generated using the official toolkit to include approximately 10 GB of data and 5,000 queries.

Decision Support Benchmark (DSB) [13]: This benchmark is developed to evaluate traditional database systems for modern decision support workloads. It is modified from the TPC-DS to include complex data distributions and challenging query templates, and it contains a total of 2,000 queries.

6.1.2 Rewrite Rules. To enhance the efficiency of the rule proposal and rewriting process for subsequent experiments, we integrate Apache Calcite [8] as our rewrite platform, alongside its comprehensive set of rewrite rules by following previous work [45]. Examples of utilized rewrite rules and their functions are illustrated in Table 1, with a complete enumeration available on the official website [1]. Specifically, we introduce a rule termed “EMPTY” to signify instances where the query remains unchanged, thereby standardizing LLM outputs with an indicator for scenarios that do not require query rewrite.

6.1.3 LLM Setting. We leverage the GPT-3.5-turbo version [10] within the ChatGPT API [2] as the default LLM setting. Furthermore, we assess our system’s generalizability across other LLMs (e.g., the leading closed-source model GPT-4 and the leading open-source models Llama3 and Granite), as detailed in Section 6.5.

6.1.4 Baseline Methods. We compare our system with two baseline methods:

Learned Rewrite (LR) [45]: This approach, recognized as the state-of-the-art query rewrite method, incorporates a cost estimation model for predicting the performance of rewritten queries. It further employs a Monte Carlo Tree-based search algorithm to identify the optimal query.

LLM only [23]: This method straightforwardly generates a rewritten query from the input, incorporating task instructions, schema, and a fixed demonstration as prompts to the LLM. When the rewritten queries are not executable or equivalent to the original queries, we use the original queries to ensure a fair comparison.

6.1.5 Training Setting. In the demonstration preparation phase, we exclude any training queries already present in the demonstration pool from being selected as demonstrations to mitigate potential bias. For the development of our query representation-based

데이터 [12]에서, 우리 모델의 훈련은 데이터 부족으로 인해 제약을 받습니다. 제한된 훈련 트리플렛에서 파생된 우리의 대조적 쿼리 튜플은 쿼리 실행의 높은 계산 비용으로 인해 확장성 문제에 직면합니다. 또한, 우리의 모델에서 쿼리 표현의 복잡성은 SimCSE에서 사용되는 단어 임베딩의 단순성을 능가합니다. 이러한 제약 조건인 제한된 데이터와 복잡한 훈련 목표를 고려하여, 우리는 교육 과정 학습 파이프라인을 채택할 것을 제안합니다. 이 접근 방식은 우리의 대조적 표현 모델의 학습 효율성과 효과를 향상하도록 설계되었습니다.

Algorithm 1 Contrastive Training under Curriculum Scheduler

```

Require: Total training data  $S_0$ , Number of iterations  $I$ 
Require: Initialized model  $E_0$ 
1:  $N_0 = \text{len}(S_0)$                                 ▶ Total number of training data
2:  $N = \lceil (N_0/I) \rceil$                           ▶ Each iteration incremental data size
3:  $Tr_0 = \emptyset$                                  ▶ Initial training data
4:  $i = 1$                                          ▶ Initial iteration
5: while  $i \leq I$  do
6:   if  $\text{len}(N) > \text{len}(S_{i-1})$  then           ▶ If less than N data left
7:      $Tr_i \leftarrow S_{i-1}$                          ▶ Select all the data left
8:      $Tr_i = Tr_{i-1} + Tr_i$                       ▶ Append to training data
9:     Train  $E_{i-1}$  on  $Tr_i$  and get  $E_i$           ▶ Continue training
10:    else
11:       $C_i \leftarrow Top_N(S_{i-1})$  based on  $conf_{E_{i-1}}(\cdot)$  ▶ Select N easy data
         following curriculum from the unvisited dataset
12:       $S_i = S_{i-1} - C_i$                          ▶ Deduct them from unvisited data
13:       $Tr_i = Tr_{i-1} + C_i$                       ▶ Append them to training data
14:      Train  $E_{i-1}$  on  $Tr_i$  and get  $E_i$           ▶ Train the model
15:       $i = i + 1$                                 ▶ Move to next iteration
16:    end if
17: end while
18: Use the final  $E_I$  for inference

```

7번 그림에 나타난 것처럼, 이 파이프라인의 본질은 전략적으로 효과적인 커리큘럼을 구현하는 것입니다. 제공된 훈련 트리플에서 시작하여, 우리는 더 작고 간단한 부분 집합에서 대조적 표현 모델을 초기 훈련하고, 남은 데이터셋에서 더 쉬운 부분 집합을 점진적으로 통합하며 모델을 재훈련시켜 모든 훈련 데이터를 활용합니다. 우리 커리큘럼을 생성하는 방법론은 알고리즘 1에 상세히 설명되어 있습니다. 이 알고리즘은 빈 모델에서 시작하며, 각 반복은 현재 모델이 가장 높은 자신감으로 수행하는 훈련 데이터의 부분 집합을 선택하고, 이 새로운 부분 집합을 통합하기 위해 모델을 재훈련하는 것을 포함합니다 (5-17줄). 이 반복적인 재훈련 과정은 전체 훈련 데이터셋이 통합될 때까지 계속됩니다.

특히, 우리는 모델이 데이터에 대한 신뢰도에 따라 남은 훈련 데이터의 더 쉬운 부분 집합을 샘플링합니다. 우리의 대조적 모델을 사용하여 두 쿼리의 임베딩을 x 과 y 로 얻는다고 가정하면, 우리는 식 3의 훈련 목표와 일관성을 유지하기 위해 코사인 유사성을 사용하여 그들의 유사도 점수를 계산할 수 있습니다. 각 대조적 쿼리 튜플 (Q, Q^+, Q^-) 에 대해, 우리는 $\text{sim}(E(Q), E(Q^+)) = 1$ 이고 $\text{sim}(E(Q), E(Q^-)) = 0$ 이기를 기대합니다. 따라서 대조적 모델의 신뢰 점수 E 를 주어진 튜플에 대해 다음과 같이 정의합니다.

$$\text{conf}_E(Q) = \text{sim}(E(Q), E(Q^+)) - \text{sim}(E(Q), E(Q^-)) + 1 \quad (4)$$

따라서, 각 반복마다, 훈련된 모델 E_{i-1} , 이전 훈련 데이터셋 T_{i-1} 및 방문하지 않은 데이터셋 D_{i-1} 을 주어, 우리는 D_{i-1} 에서 가장 높은 신뢰 점수를 가진 현재 튜플 (S_i 로 표시)을 생성할 수 있습니다. 그들은 그런 다음 훈련 세트로 이동됩니다.

새로운 훈련 세트 $T_i = T_{i-1} + S_i$ 와 방문하지 않은 새로운 데이터 세트 $D_i = D_{i-1} - S_i$ 를 생성합니다.

6 실험

이 섹션에서는 제안된 시스템의 효과성, 효율성 및 일반화 능력을 평가합니다.

6.1 실험 설정

6.1.1 데이터셋. 우리는 평가에 서로 다른 도메인에서 가져온 세 개의 데이터셋을 사용합니다:

IMDB (JOB 작업량) [21]: IMDB [25] 데이터셋은 영화, TV 프로그램, 배우에 대한 데이터로 구성됩니다. 이는 Join Order Benchmark (JOB)과 함께 복잡한 조인 쿼리 실행의 데이터베이스 관리 시스템 효율성을 테스트하는 데 사용되며, 5,000개의 쿼리를 이루어져 있습니다.

TPC-H [5]: 데이터베이스 관리 시스템을 평가하기 위한 벤치마크 데이터셋으로, 약 10GB의 데이터와 5,000개의 쿼리를 포함하도록 공식 도구를 사용하여 생성되었습니다.

의사결정 지원 벤치마크 (DSB) [13]: 이 벤치마크는 현대적인 의사결정 지원 작업 부하를 평가하기 위해 전통적인 데이터베이스 시스템을 개발하기 위해 만들어졌습니다. 이는 복잡한 데이터 분포와 도전적인 쿼리 템플릿을 포함하도록 TPC-DS를 수정하여 총 2,000개의 쿼리를 포함하고 있습니다.

6.1.2 재작성 규칙. 후속 실험을 위한 규칙 제안 및 재작성 프로세스의 효율성을 높이기 위해, 우리는 이전 연구 [45]에 따라 Apache Calcite [8]를 재작성 플랫폼으로 통합하고, 포괄적인 재작성 규칙 집합을 사용합니다. 사용된 재작성 규칙의 예와 그 기능은 표 1에 나와 있으며, 전체 목록은 공식 웹사이트 [1]에서 확인할 수 있습니다. 특히, 우리는 쿼리가 변경되지 않는 경우를 나타내는 "EMPTY"라는 규칙을 도입하여 쿼리 재작성이 필요하지 않은 시나리오에 대한 지표로 LLM 출력을 표준화합니다.

6.1.3 LLM 설정. 기본 LLM 설정으로 ChatGPT API [2] 내의 GPT-3.5-turbo 버전 [10]을 활용합니다. 또한, 제6.5조에 상세히 설명된 바와 같이 다른 LLM(예: 선도적인 폐쇄 소스 모델 GPT-4 및 선도적인 오픈 소스 모델 Llama3와 Granite)에 대한 시스템의 일반화 능력을 평가합니다.

6.1.4 기준 방법. 우리는 두 가지 기준 방법과 우리의 시스템을 비교합니다:

학습된 재작성 (LR) [45]: 이 접근 방식은 쿼리 재작성 방법론 중 최첨단으로 인정받으며, 재작성된 쿼리의 성능을 예측하기 위한 비용 추정 모델을 통합합니다. 또한 최적의 쿼리를 식별하기 위해 몬테카를로 트리 기반 검색 알고리즘을 사용합니다.

LLM만 [23]: 이 방법은 입력에서 작업 지침, 스키마 및 "고정된" 시연을 프롬프트로 LLM에 포함시켜 다시 작성된 쿼리를 직접 생성합니다. 다시 작성된 쿼리가 실행할 수 없거나 원래 쿼리와 동등하지 않은 경우 공정한 비교를 보장하기 위해 원래 쿼리를 사용합니다.

6.1.5 학습 설정. 시연 준비 단계에서, 우리는 시연 풀에 이미 존재하는 모든 학습 쿼리를 시연으로 선택되는 것에서 배제하여 잠재적인 편향을 완화합니다. 우리의 쿼리 표현 기반의 개발을 위해 $\{v^*\}$

Table 2: Execution time v.s. different query rewrite methods

Execution time(sec)	TPC-H				IMDB				DSB				
	Method	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th
Original	70.90	22.00	37.01	300.00	6.99	1.86	5.12	32.49	60.55	6.64	26.55	300.00	
LR	39.40	22.00	32.21	159.95	6.20	1.62	4.74	32.45	59.21	5.14	53.78	300.00	
LLM only (GPT-3.5)	70.67	22.00	37.01	300.00	6.96	1.86	5.10	32.49	61.60	6.53	26.40	300.00	
LLM-R² (GPT-3.5)	37.23	17.40	29.80	164.12	3.91	1.33	3.52	18.16	24.11	2.16	12.61	196.61	
% of Original	52.5%	79.1%	80.5%	54.7%	56.0%	71.3%	68.7%	55.9%	39.8%	32.5%	47.5%	65.5%	
% of LR	94.5%	79.1%	92.5%	102.6%	63.1%	82.0%	74.3%	56.0%	40.7%	42.0%	23.4%	65.5%	
% of LLM only	52.7%	79.1%	80.5%	54.7%	56.2%	71.3%	69.0%	55.9%	39.1%	33.1%	47.8%	65.5%	
LLM only (Llama3)	70.80	22.00	37.01	300.00	6.98	1.86	4.99	32.49	62.02	6.62	26.55	300.00	
LLM only (Granite)	65.06	20.73	36.25	300.00	6.03	1.85	4.91	32.49	60.55	6.64	26.55	300.00	
LLM-R² (Llama3)	38.47	18.99	31.55	161.03	5.94	1.83	4.88	29.69	51.35	6.12	27.07	300.00	
LLM-R² (Granite)	37.89	19.75	28.62	157.37	4.71	0.97	3.65	21.56	26.58	3.46	13.15	300.00	

Table 5: The average monetary cost, average number of tokens and time cost of training LLM-R².

Method	Dataset	Avg_Cost	Avg_Tk_In	Avg_Tk_Out	Training T.
LLM Only	TPC-H	0.0017	2844.16	161.54	-
	IMDB	0.0012	1371.36	49.76	-
	DSB	0.0028	5571.82	417.73	-
LLM-R ²	TPC-H	0.0006	1167.5	20.31	4061
	IMDB	0.0006	1188.36	21.35	3305
	DSB	0.0008	1299.67	7.00	6571

demonstration selector, we adopt a curriculum learning strategy encompassing four iterations ($I = 4$). Each iteration involves further training our contrastive representation model with a learning rate of 10^{-5} , a batch size of 8, over three epochs, utilizing a Tesla-V100-16GB GPU.

6.1.6 Evaluation Metrics. For the evaluation of rewrite methods, two key metrics are employed: *query execution time* and *rewrite latency*, which are respectively employed to evaluate the executing efficiency and the computational efficiency. To mitigate variability, each query is executed five times on a 16GB CPU device, with the average execution time calculated after excluding the highest and lowest values. To address the challenge posed by overly complex queries that exceed practical execution times, a maximum time limit of 300 seconds is imposed, with any query exceeding this duration assigned a default execution time of 300 seconds. This approach facilitates a broader range of experimental conditions. For assessing rewrite latency—the time required to complete a query rewrite—a custom Python script is utilized to invoke both rewrite methods, capturing the average rewrite latency across all test queries on the same hardware platform.

6.2 Executing Efficiency Evaluation

As shown in Table 2, we compare our proposed method **LLM-R²** with two baselines, documenting the mean, median, 75th, and 95th percentile execution times. The mean and median indicate general efficacy, while the 75th and 95th percentiles highlight performance in long tail cases. Our analysis reveals several key observations: (1) **LLM-R²** significantly reduces query execution time across the TPC-H, IMDB, and DSB datasets, outperforming all baseline methods. Specifically, **LLM-R²** reduces execution time to 94.5%, 63.1%, and 40.7% compared to **LR**, and to 52.7%, 56.0%, and 33.1% relative to **LLM only**, with further reductions to 52.5%, 56.0%, and 39.8% compared to the original query. This performance enhancement is due to optimized demonstration selection for prompting the LLM’s input, allowing our method to suggest superior rewrite

Table 3: The rewritten queries’ number

Method	Counts		
	Rewrite #	Improve #	Improve %
LR	258/203/ 456	192/197/193	74.42/ 97.04 /42.32
LLM only	197/102/210	68/67/8	34.5/65.68/3.81
LLM-R²	323/302/341	305/292/222	94.43/96.69/65.10

Table 4: Query execution time including the rewrite latency

Total (Latency)	TPC-H	IMDB	DSB
LR	40.98(1.58)	7.24(1.04)	60.99(1.78)
LLM only	75.37(4.70)	7.58(1.38)	64.21(6.00)
LLM-R²	40.63 (3.40)	6.81 (2.90)	27.40 (3.29)

rules. Additionally, **LLM-R²** offers more adaptable and tailored rule suggestions compared to the **LR** baseline.

(2) The bottom section of Table 2 indicates that **LLM-R²** using different LLM backbones can also outperform both **LLM Only** and **LR**, showcasing the effectiveness of our **LLM-R²** for both closed-source and open-source LLMs.

(3) The margin of improvement over **LR** is significantly greater in the IMDB and DSB datasets compared to the TPC-H dataset. This is due to two reasons: (1) Most of the effective rewrite rules for TPC-H queries can already be applied by existing methods, limiting **LLM-R²**’s potential for enhancements; (2) TPC-H’s reliance on only 22 query templates results in limited query diversity, constraining the demonstration of **LLM-R²**’s superior generalization abilities.

(4) **LR**’s underperformance on the DSB dataset is due to its greedy search algorithm. The Monte Carlo tree search in **LR** struggles with the complex and costly query trees of DSB, retaining only a few best options at each step. This limitation hampers the selection of effective rules, explaining its poor performance.

(5) **LLM only** has the worst performance. The direct generation of SQL with LLMs results in non-executable or non-equivalent rewrites, and hence many rewritten queries remain identical to the original across datasets.

Furthermore, we evaluate the performance by collecting statistics on the number of successful rewrites performed by each method across three datasets. As shown in Table 3, we observe that:

(1) **LLM-R²** excels with the most efficiency-enhancing rewrites, achieving the largest improvement percentage upon rewriting. Compared to the baseline, **LLM-R²** has both a higher number of rewrites and a significant improvement in query execution efficiency across all the evaluated datasets.

(2) **LLM only** often fails in its rewrite attempts. For instance, in the TPC-H dataset, 119 out of 129 rewrites either do not match the original query results or contain execution errors. Even in the simpler IMDB dataset, **LLM only** fails in 31 out of 102 attempts and makes limited effective rewrites due to a lack of database knowledge. In contrast, our **LLM-R²** successfully rewrites more queries and achieves a higher improvement rate across all datasets.

6.3 Computational Efficiency Evaluation

To evaluate the computational efficiency, we rigorously assess the average rewrite latency for input queries across all datasets for the **LLM-R²** framework as well as the **LR** and **LLM only** baselines. Moreover, to ascertain if query time reduction adequately compensates for the rewriting latency, we combine the execution cost and

표 2: 실행 시간 대 다양한 쿼리 재작성 방법

Method	TPC-H					IMDB					DSB				
	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th
Original	70.90	22.00	37.01	300.00	6.99	1.86	5.12	32.49	60.55	6.64	26.55	300.00			
LR	39.40	22.00	32.21	159.95	6.20	1.62	4.74	32.45	59.21	5.14	53.78	300.00			
LLM only (GPT-3.5)	70.67	22.00	37.01	300.00	6.96	1.86	5.10	32.49	61.60	6.53	26.40	300.00			
LLM-R ² (GPT-3.5)	37.23	17.40	29.80	164.12	3.91	1.33	3.52	18.16	24.11	2.16	12.61	196.61			
% of Original	52.5%	79.1%	80.5%	54.7%	56.0%	71.3%	68.7%	55.9%	39.8%	32.5%	47.5%	65.5%			
% of LR	94.5%	79.1%	92.5%	102.6%	63.1%	82.0%	74.3%	56.0%	40.7%	42.0%	23.4%	65.5%			
% of LLM only	52.7%	79.1%	80.5%	54.7%	56.2%	71.3%	69.0%	55.9%	39.1%	33.1%	47.8%	65.5%			
LLM only (Llama3)	70.80	22.00	37.01	300.00	6.98	1.86	4.99	32.49	62.02	6.62	26.55	300.00			
LLM only (Granite)	65.06	20.73	36.25	300.00	6.03	1.85	4.91	32.49	60.55	6.64	26.55	300.00			
LLM-R ² (Llama3)	38.47	18.99	31.55	161.03	5.94	1.83	4.88	29.69	51.35	6.12	27.07	300.00			
LLM-R ² (Granite)	37.89	19.75	28.62	157.37	4.71	0.97	3.65	21.56	26.58	3.46	13.15	300.00			

표 5: LLM-R2 훈련의 평균 금전적 비용, 평균 토큰 수 및 시간 비용.

Method	Dataset	Avg_Cost	Avg_Tk_In	Avg_Tk_Out	Training T.
LLM Only	TPC-H	0.0017	2844.16	161.54	-
	IMDB	0.0012	1371.36	49.76	-
	DSB	0.0028	5571.82	417.73	-
LLM-R ²	TPC-H	0.0006	1167.5	20.31	4061
	IMDB	0.0006	1188.36	21.35	3305
	DSB	0.0008	1299.67	7.00	6571

데모 선택자, 우리는 4번의 반복($I = 4$)을 포함하는 커리큘럼 학습 전략을 채택합니다. 각 반복은 학습률 10-5, 배치 크기 8, 3개의 에포크 동안 테슬라 V100 16GB GPU를 사용하여 대조적 표현 모델을 추가로 훈련하는 것을 포함합니다.

6.1.6 평가 지표. 재작성 방법의 평가를 위해 두 가지 핵심 지표가 사용됩니다: 쿼리 실행 시간과 재작성 지연 시간으로, 각각 실행 효율성과 계산 효율성을 평가하는 데 사용됩니다. 변동성을 완화하기 위해, 각 쿼리는 16GB CPU 장치에서 5번 실행되며, 최고 값과 최저값을 제외된 후 평균 실행 시간이 계산됩니다. 실용적인 실행 시간을 초과하는 지나치게 복잡한 쿼리가 제시하는 도전을 해결하기 위해, 300초의 최대 시간 제한이 부과되며, 이 시간을 초과하는 모든 쿼리는 기본 실행 시간 300초가 부여됩니다. 이 접근 방식은 더 넓은 범위의 실험 조건을 가능하게 합니다. 쿼리 재작성에 필요한 시간, 즉 재작성 지연 시간을 평가하기 위해, 사용자 정의 파이썬 스크립트가 두 가지 재작성 방법을 호출하는 데 사용되며, 이는 동일한 하드웨어 플랫폼에서 모든 테스트 쿼리에 대한 평균 재작성 지연 시간을 기록합니다.

6.2 효율성 평가 실행

표 2에서 볼 수 있듯이, 저희는 제안한 방법인 LLM-R2를 두 가지 기준선과 비교하여 평균, 중앙값, 75번째 백분위수, 95번째 백분위수 실행 시간을 기록했습니다. 평균과 중앙값은 일반적인 효능을 나타내며, 75번째와 95번째 백분위수는 긴 꼬리 사례에서의 성능을 강조합니다. 저희 분석은 몇 가지 핵심 관찰 결과를 밝혀냈습니다:

(1) LLM-R2는 TPC-H, IMDB 및 DSB 데이터셋 전반에 걸쳐 쿼리 실행 시간을 현저히 줄여 모든 기준 방법보다 우수한 성능을 보입니다. 구체적으로, LLM-R2는 LR에 비해 실행 시간을 94.5%, 63.1%, 40.7%로 줄이고, LLM에만 비해 52.7%, 56.0%, 33.1%로 줄이며, 원본 쿼리에 비해 추가로 52.5%, 56.0%, 39.8%로 줄입니다. 이러한 성능 향상은 LLM의 입력을 유도하기 위한 최적화된 데모 선택을 통해 이루어지며, 이를 통해 우리 방법이 더 우수한 재작성 제안을 할 수 있게 됩니다.

표 3: 다시 작성된 쿼리의 수

Method	Counts		
	Rewrite #	Improve #	Improve %
LR	258/203/ 456	192/197/193	74.42/ 97.04 /42.32
LLM only	197/102/210	68/67/8	34.5/65.68/3.81
LLM-R ²	323/302/341	305/292/222	94.43/96.69/65.10

표 4: 재작성 지연 시간을 포함한 쿼리 실행 시간

Total (Latency)	TPC-H	IMDB	DSB
LR	40.98(1.58)	7.24(1.04)	60.99(1.78)
LLM only	75.37(4.70)	7.58(1.38)	64.21(6.00)
LLM-R ²	40.63 (3.40)	6.81 (2.90)	27.40 (3.29)

규칙들. 또한, LLM-R2는 LR 기본선 대비 더 적응적이고 맞춤화된 규칙 제안을 제공합니다.

(2) 표 2의 하단 부분은 서로 다른 LLM 백본을 사용하는 LLM-R 2가 LLM Only와 LR 모두보다 우수한 성능을 낼 수 있음을 보여주며, 이는 우리의 LLM-R2가 폐쇄형 소스와 개방형 소스의 LM 모두에 효과적임을 나타냅니다.

(3) LR에 대한 개선 여지는 IMDB 및 DSB 데이터셋에서 TPC-H 데이터셋에 비해 현저히 큽니다. 이는 두 가지 이유로 설명할 수 있습니다: (1) TPC-H 쿼리에 대한 대부분의 효과적인 재작성 규칙은 기존 방법으로도 이미 적용할 수 있으므로 LLM-R2의 향상 잠재력이 제한됩니다; (2) TPC-H는 22개의 쿼리 템플릿에만 의존하므로 쿼리 다양성이 제한되어 LLM-R2의 뛰어난 일반화 능력을 보여주기 어렵습니다.

(4) LR의 DSB 데이터셋에서의 저조한 성과는 탐욕스러운 검색 알고리즘 때문입니다. LR의 몬테카를로 트리 탐색은 DSB의 복잡하고 비용이 많이 드는 쿼리 트리를 다루는 데 어려움을 겪으며, 각 단계에서 오직 몇 가지 최선의 옵션만 유지합니다. 이 한계는 효과적인 규칙의 선택을 방해하며, 이는 저조한 성과의 원인이 됩니다.

(5) LLM은 가장 낮은 성능만 가지고 있다. LLM을 사용하여 직접 SQL을 생성하면 실행할 수 없거나 등등하지 않은 재작성이 발생하며, 따라서 많은 재작성된 쿼리는 데이터셋 전체에서 원본과 동일하게 유지된다.

게다가, 우리는 세 개의 데이터셋에 걸쳐 각 방법이 수행한 성공적인 다시 쓰기 횟수에 대한 통계를 수집하여 성능을 평가합니다. 표 3에 표시된 것처럼, 우리는 다음과 같은 사실을 관찰합니다.

(1) LLM-R2는 가장 효율 향상에 도움이 되는 다시 쓰기 기능을 통해 우수한 성능을 발휘하며, 다시 쓰기를 통해 가장 큰 개선 비율을 달성합니다. 기준선과 비교하여 LLM-R2는 평가된 모든 데이터셋에서 더 많은 다시 쓰기 수와 쿼리 실행 효율의 상당한 향상을 모두 보여줍니다.

(2) LLM은 재작성 시도에서 자주 실패합니다. 예를 들어, TPC-H 데이터셋에서 129개의 재작성 중 119개는 원본 쿼리 결과와 일치하지 않거나 실행 오류를 포함합니다. 더 간단한 IMDB 데이터셋에서도 LLM은 102개의 시도 중 31개에서 실패하고, 데이터베이스 지식의 부족으로 인해 제한적인 효과적인 재작성을 수행합니다. 반면, 저희 LLM-R2는 모든 데이터셋에서 더 많은 쿼리를 성공적으로 재작성하고, 더 높은 개선율을 달성합니다.

6.3 계산 효율성 평가

계산 효율성을 평가하기 위해, 우리는 LLM-R2 프레임워크 및 LLM과 LLM 전용 기본선뿐 아니라 모든 데이터셋에 걸쳐 입력 쿼리에 대한 평균 재작성 지연 시간을 엄격하게 평가합니다. 또한 쿼리 시간 감소가 재작성 지연을 충분히 보상하는지 확인하기 위해 실행 비용과 $\{v^*\}$

Table 6: Execution time v.s. data scales.

Execution time(sec)	TPC-H 1G				TPC-H 5G				TPC-H 10G				TPC-H 100G			
	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th
Original	52.02	0.57	1.39	300.00	53.90	3.27	11.53	300.00	70.90	22.00	37.01	300.00	1296.85	54.35	3000.00	3000.00
LLM-R²	15.19	0.56	1.14	55.20	19.34	3.20	7.97	34.70	37.23	17.40	29.80	164.12	963.13	32.60	1330.72	3000.00
LR	25.40	0.57	1.14	213.81	20.10	4.02	9.02	32.14	39.40	22.00	37.21	159.95	1037.02	44.68	2265.75	3000.00
LLM only	52.73	2.14	4.49	300.00	54.13	3.62	11.56	300.00	70.67	22.00	37.01	300.00	1304.37	54.15	3000.00	3000.00

rewrite latency to formulate a comprehensive metric in Table 4. In order to further analyse the efficiency of **LLM-R²**, we also compute the monetary cost and **LLM-R²**’s model training time in Table 5. From the evaluations, our analysis yields significant insights: (1) **LLM-R²** incurs additional latency compared to **LR**, specifically requiring an average of 1.82, 1.86, and 1.51 seconds more to rewrite queries from the TPC-H, IMDB, and DSB datasets, respectively. This heightened latency is due to our system’s complexity. Notably, **LLM-R²** employs a demonstration selection model and leverages the online LLM API, which together account for the increased rewrite latency.

(2) However, the increased rewrite latency in our system **LLM-R²** is justifiable given that the sum of rewrite latency and execution time is lower than that of baseline methods, especially for the most complicated DSB queries. This indicates that the complex queries benefit more from our method.

(3) The **LLM only** approach exhibits considerable latency as the LLM endeavors to directly generate a rewritten query, underscoring the complexity of direct SQL query generation for LLMs. Since we have included information like table and cardinality in the demonstration selection module, **LLM-R²**’s input to LLM is much shorter than that of **LLM only**, where we also need to include the basic information like schema and query’s execution plan as inputs. The details of the input and output length difference can be found in Table 5. We also observe that this difference becomes more pronounced with the complexity of the query and database, notably in the TPC-H and DSB datasets. The comparison between our **LLM-R²** framework and the **LLM only** approach demonstrates that our methodology, which focuses on generating rewrite rules, is more effectively processed by LLMs.

(4) We also present the monetary cost and training time of our method using one Tesla-V100-16G GPU on all three datasets in seconds. As shown in Table 5, we observe that the average monetary cost per query for our **LLM-R²** is less than 0.001 USD for all datasets, which can be considered to be cost-efficient even processing large number of queries. In addition, using **LLM only** takes around 2 to 3 times of **LLM-R²**’s monetary cost. Although the average cost per query for **LLM only** is still not too high, the efficiency for using LLMs along is still much lower than our method. Moreover, training our model does not cost too much time. A single time training for less than two hours will ensure the model to be capable of dealing with all kinds of input queries querying the database.

6.4 Robustness Evaluation

We next evaluate the robustness of our **LLM-R²** framework, focusing on two critical dimensions: transferability and flexibility. Transferability evaluates the system’s ability to generalize across

Table 7: Training on TPC-H and Testing on IMDB

Execution time(sec)	Mean	Median	75th	95th
Original	6.99	1.86	5.12	32.49
LLM-R²	4.41	1.35	3.57	17.84
LR	-	-	-	-
LLM only	6.99	1.86	5.12	32.49

diverse datasets, while flexibility examines whether **LLM-R²** maintains its high performance as the volume of data increases. These aspects are crucial for understanding the adaptability and efficiency of **LLM-R²** in varied environments.

6.4.1 Transferability across different datasets. In order to evaluate our method’s transferability, we used the demonstration selection model trained on the TPC-H dataset to rewrite queries in the IMDB dataset. As shown in Table 7, the results reveal our method’s transferred performance is comparable with the in-distribution trained method and highly superior over **LLM only** when applied to a different dataset. **LLM only** fails to make effective rewrites given the fixed demonstration from the TPC-H dataset, where most rewrites lead to meaningless changes like removing table alias. Since **LR**’s cost model lacks cross-dataset transfer capability, its results are not available. These findings suggest the potential to develop a robust model by combining multiple datasets, enhancing its ability to address a wide array of unseen queries and datasets.

6.4.2 Flexibility across different data scales. To further analyze the flexibility of our method, we regenerate the TPC-H dataset using different scale factors. We additionally generate TPC-H dataset using scale factor 1 (around 1GB data), 5 (around 5GB data) and 100 (around 100GB data) apart from 10 in the main results to simulate a change of database size. From scale factor 1 to 100, we can see in Table 6 the efficiency of queries rewritten by our method increases consistently and surpasses the baseline methods, indicating the overall efficacy of our method.

6.5 Ablation Studies

We conduct an ablation study to evaluate our method’s performance along two distinct dimensions: *different selection approaches* and *specific settings in the selection model*. At first, we explore alternative selection approaches by substituting the learned selection model with different approaches to gauge their impact. Subsequently, we delve into the intricacies of the selection model by replacing individual components of the model.

6.5.1 Different selection approaches. We design the following approaches to replace the contrastive selection model in our system:

- **Zero-shot:** This method employs the LLM-R² to rewrite input queries without any preliminary demonstrations.

표 6: 실행 시간 대 데이터 규모.

Execution time(sec)	TPC-H 1G				TPC-H 5G				TPC-H 10G				TPC-H 100G				
	Method	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th
Original		52.02	0.57	1.39	300.00	53.90	3.27	11.53	300.00	70.90	22.00	37.01	300.00	1296.85	54.35	3000.00	3000.00
LLM-R ²		15.19	0.56	1.14	55.20	19.34	3.20	7.97	34.70	37.23	17.40	29.80	164.12	963.13	32.60	1330.72	3000.00
LR		25.40	0.57	1.14	213.81	20.10	4.02	9.02	32.14	39.40	22.00	37.21	159.95	1037.02	44.68	2265.75	3000.00
LLM only		52.73	2.14	4.49	300.00	54.13	3.62	11.56	300.00	70.67	22.00	37.01	300.00	1304.37	54.15	3000.00	3000.00

표 4에서 재작성 대기 시간을 포괄적인 지표로 공식화합니다. LLM-R2의 효율성을 더욱 분석하기 위해 표 5에서 통화 비용과 LLM-R2의 모델 학습 시간도 계산합니다. 평가에서 우리 분석은 중요한 통찰력을 제공합니다:

(1) LLM-R2는 LR에 비해 추가적인 지연 시간을 발생시킵니다. 특히 TPC-H, IMDB, DSB 데이터셋의 쿼리를 다시 작성하는 데 평균적으로 각각 1.82초, 1.86초, 1.51초가 더 소요됩니다. 이러한 증가된 지연 시간은 시스템의 복잡성 때문입니다. 특히 LLM-R2은 데모스트레이션 선택 모델을 사용하고 온라인 LLM API를 활용하여, 이는 다시 작성 지연 시간의 증가에 기여합니다.

(2) 그러나 우리 시스템 LLM-R2의 증가된 재작성 지연 시간은 재작성 지연 시간과 실행 시간의 합이 기본 방법보다 낮기 때문에 정당화될 수 있으며, 특히 가장 복잡한 DSB 쿼리의 경우 더욱 그렇습니다. 이는 복잡한 쿼리가 우리의 방법에서 더 많은 이점을 얻음을 나타냅니다.

(3) LLM 전용 접근 방식은 LLM이 직접 다시 작성된 쿼리를 생성하려고 시도함에 따라 상당한 지연이 발생하여 LLM에 대한 직접 SQL 쿼리 생성의 복잡성을 강조합니다. 데모 선택 모듈에 테이블 및 카디널리티와 같은 정보를 포함시켰기 때문에 LLM-R2의 LLM 입력은 LLM 전용의 입력보다 훨씬 짧습니다. 여기서 스키마 및 쿼리 실행 계획과 같은 기본 정보도 포함해야 합니다. 입력 및 출력 길이 차이에 대한 자세한 내용은 표 5를 참조하십시오. 또한 쿼리와 데이터베이스가 복잡해질수록 이 차이가 더 두드러진다는 것을 알 수 있습니다. 특히 TPC-H 및 DSB 데이터셋에서 그렇습니다. LLM-R2 프레임워크와 LLM 전용 접근 방식의 비교는 재작성 규칙 생성에 초점을 맞춘 우리의 방법론이 LLM에 의해 더 효과적으로 처리된다는 것을 보여줍니다.

(4) 또한 우리는 테슬라 V100-16G GPU 하나를 사용하여 세 가지 데이터셋 모두에서 우리 방법의 금전적 비용과 훈련 시간을 초 단위로 제시합니다. 표 5에서 볼 수 있듯이, 우리 LLM-R2의 쿼리당 평균 금전적 비용은 모든 데이터셋에서 0.001 USD 미만으로, 많은 수의 쿼리를 처리하더라도 비용 효율적이라고 볼 수 있습니다. 또한, LLM만 사용하는 것은 LLM-R2의 금전적 비용의 약 2~3배 정도만 소요됩니다. LLM만 사용하는 경우의 쿼리 당 평균 비용이 여전히 너무 높지는 않지만, LLM만을 사용하는 효율성은 우리 방법보다 여전히 훨씬 낮습니다. 게다가, 우리 모델을 훈련시키는 데는 시간이 많이 걸리지 않습니다. 2시간 미만의 단일 훈련 시간만으로도 모델이 데이터베이스에 대한 모든 종류의 입력 쿼리를 처리할 수 있도록 보장합니다.

6.4 견고성 평가

다음으로 우리는 LLM-R2 프레임워크의 견고함을 평가하며, 두 가지 중요한 차원인 전이 가능성과 {v*}유연성에 초점을 맞춥니다. 전이 가능성은 시스템이 다양한 상황에 일반화할 수 있는 능력을 평가합니다.

표 7: TPC-H로 학습하고 IMDB로 테스트

Execution time(sec)	Mean	Median	75th	95th
Original	6.99	1.86	5.12	32.49
LLM-R ²	4.41	1.35	3.57	17.84
LR	-	-	-	-
LLM only	6.99	1.86	5.12	32.49

다양한 데이터셋, 반면 {v*}유연성은 LLM-R2가 데이터 양이 증가함에 따라 높은 성능을 유지하는지 여부를 조사합니다. 이러한 측면은 다양한 환경에서 LLM-R2의 적응성과 효율성을 이해하는 데 중요합니다.

6.4.1 다른 데이터셋 간의 전이 가능성. 우리 방법의 전이 가능성은 평가하기 위해, 우리는 TPC-H 데이터셋에서 훈련된 시연 선택 모델을 사용하여 IMDB 데이터셋의 쿼리를 다시 작성했습니다. 표 7에 표시된 바와 같이, 결과는 우리 방법의 전이된 성능이 인분포 훈련 방법과 비교할 만하고 LLM에만 적용될 때 다른 데이터셋에서 훨씬 우수하다는 것을 보여줍니다. LLM은 TPC-H 데이터셋의 고정된 시연을 주어 효과적인 다시 작성성을 실패하며, 대부분의 다시 작성은 테이블 별칭 제거와 같은 무의미한 변경으로 이어집니다. LR의 비용 모델은 교차-데이터셋 전이 능력이 부족하여 그 결과가 사용 불가능합니다. 이러한 결과는 여러 데이터셋을 결합하여 강력한 모델을 개발할 잠재력을 시사하며, 이는 보지 못한 다양한 쿼리와 데이터셋을 처리할 수 있는 능력을 향상시킵니다.

6.4.2 다양한 데이터 규모에 대한 유연성. 우리의 방법의 유연성을 더욱 분석하기 위해, 우리는 다른 스케일 팩터를 사용하여 TPC-H 데이터셋을 재생성합니다. 우리는 또한 주 결과에 있는 10 이외에 스케일 팩터 1(약 1GB 데이터), 5(약 5GB 데이터) 및 100(약 100GB 데이터)을 사용하여 TPC-H 데이터셋을 생성하여 데이터베이스 크기의 변화를 시뮬레이션합니다. 스케일 팩터 1에서 100까지, 표 6에서 볼 수 있듯이, 우리의 방법에 의해 다시 작성된 쿼리의 효율성은 일관되게 증가하고 기준 방법보다 뛰어나며, 이는 우리의 방법의 전반적인 효과를 나타냅니다.

6.5 제거 연구

우리는 두 가지 다른 차원, 즉 다른 선택 접근 방식과 선택 모델의 특정 설정에서 우리의 방법의 성능을 평가하기 위해 제거 연구를 수행합니다. 먼저, 학습된 선택 모델을 다른 접근 방식으로 대체하여 그 영향을 측정하기 위해 대체 선택 접근 방식을 탐색합니다. 그 다음, 우리는 모델의 개별 구성 요소를 대체하여 선택 모델의 세부 사항을 심층적으로 조사합니다.

6.5.1 다른 선택 접근법. 우리는 시스템의 대조적 선택 모델을 대체하기 위해 다음과 같은 접근법을 설계합니다: - 제로샷: 이 방법은 LLM-R2를 사용하여 예비 데모 없이 입력 쿼리를 다시 작성합니다.

Table 8: Execution time v.s. different selection approaches.

Execution time	TPC-H				IMDB				DSB			
	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th
Original	70.90	22.00	37.01	300.00	6.99	1.86	5.12	32.49	60.55	6.64	26.55	300.00
Zero-shot	46.15	21.95	33.26	300.00	6.98	1.85	5.12	32.49	34.53	3.35	11.52	300.00
Random	40.50	21.63	32.22	165.63	5.45	1.70	4.50	25.03	45.88	5.43	17.41	300.00
Tree	39.21	18.97	30.89	164.10	4.40	1.24	3.40	18.89	26.10	3.86	13.54	240.74
SentTrans	40.19	19.21	32.21	164.99	6.05	1.70	4.49	30.01	24.68	3.95	13.18	197.23
LLM-R²	37.23	17.40	29.80	164.12	3.91	1.33	3.52	18.16	24.11	2.16	12.61	196.61

- **Few-shots:** Building on insights from Section 4, we refine the demonstration pool with three intuitive methods for one-shot demonstration selection: **Random**, **Tree**, and **SentTrans**.

As shown in Table 8 we make the following observations:

(1) **Effectiveness of the LLM-enhanced system:** The Zero-shot approach outperforms the original queries significantly, which indicates that the LLM-R² component within our rewrite system is capable of enhancing original queries, showcasing the underlying potential of the LLM to offer viable query rewrite suggestions.

(2) **Effectiveness of introducing demonstrations:** We observe that approaches incorporating demonstrations into the rewrite system consistently surpass the Zero-shot setting across all datasets. This observation underscores the significance of leveraging demonstrations to enhance the rewrite system. Furthermore, the improvement across diverse datasets highlights the universal applicability and effectiveness of demonstration-based prompting in refining rewrite outcomes.

(3) **Effectiveness of the contrastive selection model:** Our comparative analysis underscores the significance of selecting high-quality demonstrations for query rewriting. The findings reveal that superior demonstrations directly contribute to the generation of more effective rewritten queries.

6.5.2 Effectiveness of specific settings in the selection model. In this experiment, we assess three critical aspects of the contrastive selection model:

- **The Curriculum Learning pipeline:** We compare the efficacy of the curriculum learning pipeline with a baseline model trained on the TPC-H dataset using all training triplets simultaneously.

- **Demonstration Quantity:** We evaluate the impact of varying the number of demonstrations, focusing on 1-shot and 3-shot configurations to understand their effect on model performance.

- **Different LLMs:** We explore the integration of GPT-4, Llama3-8B [32] and Granite [26] into our rewriting system. Due to the cost of the GPT-4 API, we limit its use to the test dataset rewrite process, using GPT-3.5-turbo for demonstrations and models.

Table 9 shows the evaluation results and we obtain the following key insights:

(1) Our query representation model outperforms the baseline approaches in selecting optimal demonstrations, especially when curriculum-based training is adopted. Direct training on the full dataset reduces execution costs by an average of 32.17 seconds and a median of 2.3 seconds. Curriculum learning further enhances efficiency, with average reductions of 1.5 seconds and median decreases of 2.3 seconds. These results underscore the efficacy of our proposed query representation model and the curriculum learning framework.

Table 9: Ablation on curriculum, number of shots and LLM backbone.

Execution time(sec)	Mean	Median	75th	95th
LLM-R² (1-shot)	37.23	17.40	29.80	164.12
w/o Curriculum	38.73	19.70	32.17	164.98
LLM-R² (3-shots)	54.08	19.67	37.01	300.00
LLM-R² (GPT-4)	38.58	20.32	32.27	167.26
LLM-R² (Llama3)	38.47	18.99	31.55	161.03
LLM-R² (Granite)	37.89	19.75	28.62	157.37

Table 10: Ablation on different information factors sorted by Mean value.

TPC-H	Execution time		Counts
	Method	Mean/Median/75th/95th	Total/ Improve/%
LLM-R²	37.23/17.40/29.80/164.12		323/305/94.43
LR	39.40/22.00/32.21/ 159.95		258/192/74.42
Full_schema+Value	38.09/19.08/30.82/160.19		334/313/93.71
Filtered_schema	50.36/1.27/32.66/300.00		178/155/87.08
Full_schema	50.52/20.56/32.61/300.00		179/158/88.27
Filtered_schema+Value	53.81/20.73/31.53/300.00		222/198/89.19
Full_schema+Value+Plan	54.93/22.00/37.01/300.00		135/121/89.63
Full_schema+Plan	55.24/22.00/37.01/300.00		119/93/78.15
Full_schema+Card	56.23/22.00/37.01/300.00		56/33/58.93
Plan	57.20/22.00/7.01/300.00		44/42/95.45
Full_schema+Value+Card	68.67/22.00/37.01/300.00		48/43/89.58
Card	70.90/22.00/37.01/300.00		0/0/-

(2) Using a 3-shot approach instead of 1-shot degrades performance. The 3-shot method generated only 255 rewrite proposals, with 235 improving query execution efficiency, despite a 92.16% success rate. The reduced number of suggestions is due to inconsistent guidance from three demonstrations, higher rewrite costs, and longer in-context texts for LLM analysis. Thus, 1-shot prompting is more efficient and effective under current conditions.

(3) we use the leading Llama3 from the Llama family [32] and Granite [26] as the LLM backbones for both our LLM-R² and the baseline LLM Only method. We evaluate them on all three datasets and record the results in Table 2. We observe that changing the LLM backbone into Llama3 or Granite decreases both LLM-R²'s and LLM Only's performance. Moreover, the performance of LLM-R² using Llama3 and Granite still outperforms all the baselines, showcasing the effectiveness of our method.

(4) Despite GPT-4's enhanced capabilities and Llama3 or Granite's strong instruction-following ability, transitioning to a different model for inference adversely impacts the efficacy of our method. This observation underscores the complexity of optimizing performance within our proposed framework and suggests that consistency in model usage throughout the process may be pivotal for achieving optimal selection.

6.5.3 Additional Information in LLM Prompts. Including data distribution and cardinality factors in demonstrations and the inputs of LLM is also worth discussing as they are widely adopted in txt-to-SQL methods [6, 23]. We design the following information that can be included in demonstrations and the LLM's inputs together with the queries as another ablation study:

Full_schema: The information of the database schema, including table names, column names and column types;

Filtered_schema: The filtered information of the database schema, where only the tables and columns queried are included;

표 8: 실행 시간 대 다양한 선택 접근 방식.

Execution time	TPC-H				IMDB				DSB			
	Mean	Median	75th	95th	Mean	Median	75th	95th	Mean	Median	75th	95th
Original	70.90	22.00	37.01	300.00	6.99	1.86	5.12	32.49	60.55	6.64	26.55	300.00
Zero-shot	46.15	21.95	33.26	300.00	6.98	1.85	5.12	32.49	34.53	3.35	11.52	300.00
Random	40.50	21.63	32.22	165.63	5.45	1.70	4.50	25.03	45.88	5.43	17.41	300.00
Tree	39.21	18.97	30.89	164.10	4.40	1.24	3.40	18.89	26.10	3.86	13.54	240.74
SentTrans	40.19	19.21	32.21	164.99	6.05	1.70	4.49	30.01	24.68	3.95	13.18	197.23
LLM-R ²	37.23	17.40	29.80	164.12	3.91	1.33	3.52	18.16	24.11	2.16	12.61	196.61

- 소샘: 제4장의 통찰력을 바탕으로, 우리는 세 가지 직관적인 방법(무작위, 트리, SentTrans)을 사용하여 소샘 풀을 정제하여 한 번만 소샘 선택을 합니다.

표 8에서 알 수 있듯이 다음과 같은 관찰을 할 수 있습니다:

(1) LLM 강화 시스템의 효과: 제로샷 접근 방식이 원본 쿼리보다 훨씬 우수하여, 우리 재작성 시스템 내의 LLM-R2 구성 요소가 원본 쿼리를 향상시킬 수 있음을 나타내며, 이는 LLM이 실행 가능한 쿼리 재작성 제안을 제공할 수 있는 잠재력을 보여줍니다.

(2) 시연 도입의 효과: 우리는 시연을 재작성 시스템에 통합하는 접근 방식이 모든 데이터셋에서 제로샷 설정보다 일관되게 우수하다는 것을 관찰합니다. 이 관찰은 시연을 활용하여 재작성 시스템을 향상시키는 중요성을 강조합니다. 또한 다양한 데이터셋 전반에 걸친 개선 사항은 시연 기반 프롬프트가 재작성 결과 개선에 보편적으로 적용 가능하고 효과적임을 보여줍니다.

(3) 대조적 선택 모델의 효과: 우리의 비교 분석은 쿼리 재작성을 위한 고품질 시연 선택의 중요성을 강조합니다. 연구 결과는 우수한 시연이 더 효과적인 재작성 쿼리 생성에 직접적으로 기여함을 보여줍니다.

6.5.2 선택 모델에서 특정 설정들의 효과. 이 실험에서는 대조적 선택 모델의 세 가지 중요한 측면을 평가합니다:

- 커리큘럼 학습 파이프라인: 우리는 TPC-H 데이터셋을 사용하여 모든 훈련 트리플을 동시에 학습한 기본 모델과 커리큘럼 학습 파이프라인의 효과를 비교합니다.

- 시연 양: 우리는 시연 횟수를 변경하여 그 영향을 평가하며, 1-샷 및 3-샷 구성에 초점을 맞추어 모델 성능에 미치는 영향을 이해합니다.

- 다양한 LLM: 우리는 GPT-4, Llama3-8B [32] 및 Granite [26]를 리라이팅 시스템에 통합하는 것을 탐구합니다. GPT-4 API의 비용으로 인해, 우리는 테스트 데이터셋 리라이팅 과정에 그 사용을 제한하고, 데모 및 모델을 위해 GPT-3.5-turbo를 사용합니다.

표 9는 평가 결과를 보여주며, 우리는 다음과 같은 주요 통찰력을 얻었습니다:

(1) 우리의 질의 표현 모델은 최적의 시연을 선택하는 데 있어 기준 접근법보다 우수한 성능을 보이며, 특히 커리큘럼 기반 훈련이 채택될 때 더욱 그렇다. 전체 데이터셋에 대한 직접 훈련은 평균 32.17초, 중앙값 2.3초의 실행 비용을 줄인다. 커리큘럼 학습은 평균 1.5초의 감소와 중앙값 2.3초의 감소를 통해 효율성을 더욱 향상시킨다. 이러한 결과는 우리가 제안한 질의 표현 모델과 커리큘럼 학습 프레임워크의 효과를 강조한다.

표 9: 커리큘럼, 샷 수 및 LLM 백본에 대한 제거 실험.

Execution time(sec)	Mean	Median	75th	95th
LLM-R² (1-shot)	37.23	17.40	29.80	164.12
w/o Curriculum	38.73	19.70	32.17	164.98
LLM-R² (3-shots)	54.08	19.67	37.01	300.00
LLM-R² (GPT-4)	38.58	20.32	32.27	167.26
LLM-R² (Llama3)	38.47	18.99	31.55	161.03
LLM-R² (Granite)	37.89	19.75	28.62	157.37

표 10: 평균 값에 따라 정렬된 다양한 정보 요소의 제거 실험.

TPC-H	Execution time		Counts
	Method	Mean/Median/75th/95th	Total/ Improve/%
LLM-R²	37.23/17.40/29.80/164.12		323/305/94.43
LR	39.40/22.00/32.21/ 159.95		258/192/74.42
Full_schema+Value	38.09/19.08/30.82/160.19		334/313/93.71
Filtered_schema	50.36/1.27/32.66/300.00		178/155/87.08
Full_schema	50.52/20.56/32.61/300.00		179/158/88.27
Filtered_schema+Value	53.81/20.73/31.53/300.00		222/198/89.19
Full_schema+Value+Plan	54.93/22.00/37.01/300.00		135/121/89.63
Full_schema+Plan	55.24/22.00/37.01/300.00		119/93/78.15
Full_schema+Card	56.23/22.00/37.01/300.00		56/33/58.93
Plan	57.20/22.00/7.01/300.00		44/42/95.45
Full_schema+Value+Card	68.67/22.00/37.01/300.00		48/43/89.58
Card	70.90/22.00/37.01/300.00		0/0/-

(2) 1샷 접근 방식 대신 3샷 접근 방식을 사용하는 것은 성능을 저하합니다. 3샷 방법은 255개의 재작성 제안만 생성했으며, 그중 235개가 쿼리 실행 효율성을 향상시켰습니다. 이는 92.16%의 성공률에도 불구하고 세 가지 시연의 일관되지 않은 안내, 더 높은 재작성 비용, 그리고 LLM 분석을 위한 더 긴 맥락 텍스트 때문입니다. 따라서 현재 상황에서는 1샷 프롬프트가 더 효율적이고 효과적입니다.

(3) 우리는 Llama 가족 [32]에서 선도적인 Llama3와 Granite [26]를 우리의 LLM-R2와 기본 LLM Only 방법 모두의 LLM 백본으로 사용합니다. 우리는 세 가지 데이터셋 모두에서 이를 평가하고 표 2에 결과를 기록합니다. 우리는 LLM 백본을 Llama3 또는 Granite로 변경하면 LLM-R2와 LLM Only의 성능이 모두 저하되는 것을 관찰합니다. 게다가, Llama3와 Granite를 사용하는 LL M-R2의 성능은 여전히 모든 기본값을 능가하여 우리의 방법의 효과를 보여줍니다.

(4) GPT-4의 향상된 기능과 Llama3 또는 Gran-it의 강력한 지침 준수 능력에도 불구하고, 추론에 다른 모델로 전환하는 것은 우리의 방법의 효과에 부정적인 영향을 미칩니다. 이 관찰은 제안된 프레임워크 내에서 성능 최적화의 복잡성을 강조하며, 프로세스 전반에 걸쳐 모델 사용의 일관성이 최적의 선택을 달성하는 데 핵심적일 수 있음을 시사합니다.

6.5.3 LLM 프롬프트의 추가 정보. 시연 및 LLM의 입력에 데이터 분포 및 카디널리티 요소를 포함시키는 것은 텍스트-투-SQL 방법에서 널리 채택되고 있으므로 [6, 23] 논의할 가치가 있습니다. 우리는 다음과 같은 정보를 시연 및 LLM의 입력에 포함시켜 쿼리와 함께 또 다른 제거 연구로 설계했습니다:

전체 스키마: 데이터베이스 스키마의 정보, 테이블 이름, 열 이름 및 열 유형 포함;

필터링된 스키마: 데이터베이스 스키마의 필터링된 정보로, 쿼리된 테이블과 열만 포함됨;

Original Query	Original Query	Original Query
Original query: select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = 2-HIGH then 1 else 0 end) as high_line_count, sum(case when o_orderpriority > '1-URGENT' and o_orderpriority < '2-HIGH' then 1 else 0 end) as low_line_count from orders, lineitem where o_orderkey = l_orderkey and ... group by l_shipmode order by l_shipmode; Query cost: 21.63845668	Original query: select s_acctbal, ..., s_comment from part, supplier, ..., region where p_partkey = ps_partkey and ... and ps_supplycost = (select min(ps_supplycost) from partsupp, supplier, nation, region where p_partkey = ps_partkey and ... and r_name = 'AMERICA') order by s_acctbal desc, ..., p_partkey; Query cost: 0.78970573	Original query: select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriority from customer, orders, lineitem where c_mktsegment = 'AUTOMOBILE' and ... group by l_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate; Query cost: 33.2621007
LR Result Rules applied: []	LR Result Rules applied: [AGGREGATE_JOIN_TRANSPOSE, SORT_PROJECT_TRANSPOSE, JOIN_EXTRACT_FILTER] New query cost: 17.40275009	LR Result Rules applied: [FILTER_INTO_JOIN, PROJECT_TO_CALC, JOIN_EXTRACT_FILTER] New query cost: 33.1077284
QueryCL Result Rules applied: [FILTER_INTO_JOIN] New query cost: 17.65787496	QueryCL Result Rules applied: []	QueryCL Result Rules applied: [FILTER_INTO_JOIN, JOIN_EXTRACT_FILTER, PROJECT_TO_CALC] New query cost: 26.90622425

Figure 8: Examples of the rewrite results of baseline Learned Rewrite method and our LLM-R² method.

Table 11: The variety of rules in terms of unique rules and total applications.

Counts	TPC-H		IMDB		DSB	
	Method	Unique	Total	Method	Unique	Total
LR	5	405	1	192	9	707
LLM-R ²	56	1824	6	361	37	920

Value: To let the LLMs understand the data distribution and value range information, we follow existing txt-to-SQL work [6, 23] to include value descriptions to each column of the database schema;

Plan: We directly include the physical plan obtained from DBMS on the query as a query tree ;

Card: We specially extract the cardinality information from the physical plans and formalize a cardinality tree as input.

The results of the experiment are shown in Table 10. From the results we have a few observations.

(1) Adding more information to demonstrations and LLM inputs does not help the LLM generate better rewrite suggestions. Our LLM-R² outperforms all the other combination of additional information. We believe that the Occam’s Razor also exists in LLMs, especially for dealing with complex task like SQL.

(2) Current LLMs still cannot handle structural information well when directly given such information. When we include Plan or Card in the inputs, which are in the tree format, the rewrite performance decreases significantly. The performance drop is mainly due to much fewer rewrite suggestions, where the LLM no longer considers some queries as ‘need to rewrite and improve’. In the extreme case where we only include a tree of cardinality values, none of the input queries are rewritten.

(3) The tuning of the LLM prompt is sensitive and have no obvious pattern. For example, ‘Full_schema+Value’ can achieve a comparable performance as LLM-R², but using the filtered schema instead results in a much lower performance. This observation is in favor of the findings in [30], that LLMs are sensitive to their prompts.

6.6 Qualitative Analysis

we proceed to present examples to illustrate the rewrite quality between various methods, focusing particularly on comparisons between our approach and baseline methods. Notably, due to the high incidence of erroneous rewrites generated by the **LLM-only** method, our analysis primarily compares our method against the **LR** baseline. Figure 8 demonstrates our findings demonstrate the superior robustness and flexibility of our model compared to **LR**. For instance, in the first case study, our **LLM-R²** method uncovers rewrite rules that remain undetected by **LR**. This discrepancy can be attributed to **LR**’s potentially ineffective cost model, which might erroneously consider the original query as already optimized.

Conversely, our LLM-enhanced system suggests a rewrite that evidences significant potential for cost reduction. In the second case, **LR** is observed to occasionally transform an efficient query into a less efficient one. In the third scenario, **LLM-R²** outperforms by modifying the rule sequence and incorporating an additional “**FILTER_INTO_JOIN**” operation, transforming a “WHERE” clause into an “INNER JOIN”, thereby achieving a more efficient query rewrite than that offered by **LR**.

Furthermore, we delve into the diversity of rewrite rules suggested by the different methods. Here, the term *Unique* refers to the distinct categories of rewrite rules recommended by a method, whereas *Total* denotes the aggregate count of all rewrite rule instances proposed. As illustrated in Table 11, it is evident that **LLM-R²** not only recommends a higher quantity of rewrite rules but also exhibits a broader spectrum of rewrite strategies by employing a diverse range of rules. This observation underscores **LLM-R²**’s enhanced flexibility and robustness, showcasing its capability to generate more varied and effective rewrite plans.

7 CONCLUSION

In this paper, we propose a LLM-enhanced query rewrite pipeline to perform efficient query rewrite. By collecting useful demonstrations and learning a contrastive demonstration selector to modify the rewrite system inputs, we are able to successfully improve the input queries’ efficiency across popular datasets. In addition, we further prove the effectiveness of our learning pipeline and the transferability of our method over different scales, model backbones, and datasets, showing that LLM-enhanced methods could be an effective solution for efficiency-oriented query rewrite. The current limitation is that our LLM-R² exhibits higher rewrite latency compared to DB-only methods due to the time consumed by calling LLM APIs and selecting demonstrations. However, our experimental results show that this increased latency is offset by the larger reduction in execution time achieved by LLM-R². This demonstrates the potential of LLMs in database applications, leveraging their strong generalization and reasoning capabilities. Future improvements could include efficient demonstration selection algorithms like Faiss[14] or fine-tuning an LLM specifically for query rewriting with more datasets.

ACKNOWLEDGMENTS

This research is supported, in part, by Alibaba Group through Alibaba Innovative Research (AIR) Program and Alibaba-NTU Singapore Joint Research Institute (JRI), and the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Awards MOE-T2EP20221-0015 and MOE-T2EP20223-0004).

Original Query	Original Query	Original Query
Original query: select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = 2-HIGH then 1 else 0 end) as high, l_inx_count, sum(case when o_orderpriority > '1-URGENT' and o_orderpriority < '2-HIGH' then 1 else 0 end) as low, l_inx_count from orders, lineitem where o_orderkey = l_orderkey and ... group by l_shipmode order by l_shipmode; Query cost: 21.6384566	Original query: select s_acctbal, ..., s_comment from part, supplier, ..., region where p_partkey = ps_partkey and ... and ps_supplycost = (select min(ps_supplycost) from partsupp, supplier, nation, region where p_partkey = ps_partkey and ... and r_name = 'AMERICA') order by s_acctbal desc, ..., p_partkey; Query cost: 0.78970533	Original query: select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdate, o_shippriority from customer, orders, lineitem where c_mktsegment = 'AUTOMOBILE' and ... group by l_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate; Query cost: 33.2621007
LR Result Rules applied: []	LR Result Rules applied: [AGGREGATE_JOIN_TRANSPOSE, SORT_PROJECT_TRANSPOSE, JOIN_EXTRACT_FILTER] New query cost: 17.40275009	LR Result Rules applied: [FILTER_INTO_JOIN, PROJECT_TO_CALC, JOIN_EXTRACT_FILTER] New query cost: 33.1077284
QueryCL Result Rules applied: [FILTER_INTO_JOIN] New query cost: 17.65787496	QueryCL Result Rules applied: []	QueryCL Result Rules applied: [FILTER_INTO_JOIN, JOIN_EXTRACT_FILTER, PROJECT_TO_CALC] New query cost: 26.90622425

그림 8: 기본 Learned Rewrite 방법과 당사의 LLM-R2 방법의 재작성 결과 예시.

표 11: 고유 규칙과 전체 적용 측면에서 규칙의 다양성.

Counts	TPC-H		IMDB		DSB		
	Method	Unique	Total	Unique	Total	Unique	Total
LR	5	405	1	192	9	707	
LLM-R ²	56	1824	6	361	37	920	

값: LLM이 데이터 분포와 값 범위 정보를 이해할 수 있도록 기존 텍스트-투-SQL 작업 [6, 23]을 따라 데이터베이스 스키마의 각 열에 값 설명을 포함합니다; 계획: 쿼리 트리 형태로 DBMS에서 얻은 물리적 계획을 쿼리에 직접 포함합니다;

카드: 우리는 물리적 계획에서 특히 카디널리티 정보를 추출하고 카디널리티 트리를 입력으로 형식화합니다.

실험 결과는 표 10에 나와 있습니다. 결과에서 몇 가지 관찰 사항을 얻을 수 있습니다.

(1) 시연 및 LLM 입력에 더 많은 정보를 추가하는 것은 LLM이 더 나은 다시 쓰기 제안을 생성하는 데 도움이 되지 않습니다. 우리의 LLM-R2는 추가 정보의 다른 모든 조합보다 우수한 성능을 발휘합니다. 우리는 특히 SQL과 같은 복잡한 작업을 처리하는데 있어서도 LLM에 오컴의 면도날이 존재한다고 믿습니다.

(2) 현재 LLM(대규모 언어 모델)은 직접 구조적 정보를 제공받을 때 여전히 이를 잘 처리하지 못합니다. 계획 또는 카드와 같은 트리 형식의 정보를 입력에 포함하면 다시 작성 성능이 현저히 감소합니다. 성능 저하는 주로 다시 작성 제안이 훨씬 적기 때문이며, LLM은 더 이상 일부 쿼리를 '다시 작성하고 개선해야 함'으로 간주하지 않습니다. 카드에 포함된 정보가 오직 기수(cardinality) 값의 트리인 극단적인 경우, 입력 쿼리 중 어느 것도 다시 작성되지 않습니다.

(3) LLM 프롬프트 조정은 민감하고 명확한 패턴이 없습니다. 예를 들어, 'Full_schema+Value'는 LLM-R2와 유사한 성능을 낼 수 있지만, "필터링된 스키마"를 사용하면 성능이 훨씬 저하됩니다. 이 관찰은 [30]의 연구 결과와 일치하며, LLMs가 프롬프트에 민감하다는 것을 보여줍니다.

6.6 질적 분석

우리는 다양한 방법들 사이의 다시 쓰기 품질에 대한 예를 제시하여 설명하는 것으로 진행하며, 특히 우리의 접근 방식과 기본 선 방법들 사이의 비교에 초점을 맞춥니다. 특히, LLM 전용 방법에 의해 생성된 오류 다시 쓰기의 높은 발생률로 인해, 우리의 분석은 주로 우리의 방법과 LR 기본선을 비교합니다. 그림 8은 우리의 발견이 LR에 비해 우리의 모델의 뛰어난 견고성과 유연성을 보여준다는 것을 보여줍니다. 예를 들어, 첫 번째 사례 연구에서, 우리의 LLM-R2 방법은 LR에 의해 미검출된 다시 쓰기 규칙을 밝혀냅니다. 이 불일치는 LR의 잠재적으로 비효과적인 비용 모델로 인해 발생할 수 있으며, 이는 원래 쿼리가 이미 최적화되었다고 오류로 간주할 수 있습니다.

반대로, 우리 LLM 기반 시스템은 비용 절감 가능성이 크게 나타나는 재작성을 제안합니다. 두 번째 경우, LR은 때때로 효율적인 쿼리를 덜 효율적인 것으로 변환하는 것이 관찰됩니다. 세 번째 시나리오에서 LLM-R2는 규칙 순서를 수정하고 추가 "FILTER_INTO_JOIN" 작업을 통합하여 "WHERE" 절을 "INNER JOIN"으로 변환함으로써 LR이 제공하는 것보다 더 효율적인 쿼리 재작성을 달성합니다.

게다가, 우리는 다양한 재작성 규칙을 제안하는 서로 다른 방법들의 다양성을 탐구합니다. 여기서 "유일한"이라는 용어는 한 방법이 권장하는 독특한 재작성 규칙의 범주를 의미하는 반면, "전체"는 모든 재작성 규칙 인스턴스의 총 개수를 나타냅니다. 표 11에서 볼 수 있듯이, LLM-R2는 더 많은 재작성 규칙을 권장할 뿐만 아니라 다양한 규칙을 사용하여 더 넓은 재작성 전략의 스펙트럼을 보여줍니다. 이 관찰은 LLM-R2의 향상된 유연성과 견고함을 강조하며, 더 다양하고 효과적인 재작성 계획 생성 능력을 보여줍니다.

7 결론

이 논문에서는 효율적인 쿼리 재작성을 수행하기 위해 LLM(대규모 언어 모델)을 활용한 쿼리 재작성 파이프라인을 제안합니다. 유용한 시연을 수집하고 대조적 시연 선택기를 학습하여 재작성 시스템의 입력을 수정함으로써, 다양한 인기 있는 데이터 셋에서 입력 쿼리의 효율성을 성공적으로 향상시킬 수 있었습니다. 또한, 우리는 학습 파이프라인과 방법의 전이 가능성성을 증명하고, 서로 다른 규모, 모델 백본, 데이터셋에 적용할 수 있음을 보였습니다. 이는 LLM을 활용한 방법이 효율성 지향 쿼리 재작성을 위한 효과적인 해결책이 될 수 있음을 나타냅니다. 현재 제한점은 LLM-R2가 DB 전용 방법에 비해 LLM API 호출과 시연 선택에 소요되는 시간으로 인해 재작성 지연 시간이 더 높다는 것입니다. 그러나 실험 결과, LLM-R2에 의해 달성된 더 큰 실행 시간 감소가 이 증가된 지연 시간을 상쇄함을 알 수 있었습니다. 이는 데이터베이스 애플리케이션에서 LLMs의 잠재력을 보여주며, 그들의 강력한 일반화 및 추론 능력을 활용합니다. 향후 개선 사항에는 Faiss[14]와 같은 효율적인 시연 선택 알고리즘이나 쿼리 재작성을 위해 더 많은 데이터셋으로 LLM을 미세 조정하는 것이 포함될 수 있습니다.

감사의 말

이 연구는 알리바바 그룹의 알리바바 혁신 연구(AIR) 프로그램과 알리바바-NTU 싱가포르 공동 연구소(JRI), 그리고 싱가포르 교육부 학술 연구 기금(Tier 2 Awards MOE-T2EP2021-0015 및 MOE-T2EP2022-0004)의 지원을 받아 진행되었습니다.

REFERENCES

- [1] [n.d.] Apache Calcite Rewrite Rules. <https://calcite.apache.org/javadocAggregate/org/apache/calcite/rel/rules/package-summary.html>.
- [2] [n.d.] Introduction of OpenAI Text Generation APIs. <https://platform.openai.com/docs/guides/text-generation>.
- [3] [n.d.] LLM As Database Administrator. <https://github.com/TsinghuaDatabaseGroup/DB-GPT>.
- [4] [n.d.] PostgreSQL. <https://www.postgresql.org>.
- [5] [n.d.] TPC-H Toolkit. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp.
- [6] Arian Askari, Christian Poelitz, and Xinye Tang. 2024. MAGIC: Generating Self-Correction Guideline for In-Context Text-to-SQL. *CoRR* abs/2406.12692 (2024).
- [7] Qiuishi Bai, Sadeem Alsudais, and Chen Li. 2023. QueryBooster: Improving SQL Performance Using Middleware Services for Human-Centered Query Rewriting. *Proc. VLDB Endow.* 16, 11 (2023), 2911–2924.
- [8] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In *SIGMOD*. 221–230.
- [9] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML*, Vol. 382. 41–48.
- [10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. In *NeurIPS*.
- [11] Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. 2021. Curriculum Labeling: Revisiting Pseudo-Labeling for Semi-Supervised Learning. In *AAAI*. 6912–6920.
- [12] Daniel M. Cer, Mona T. Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. [n.d.] SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *ACL*. 1–14.
- [13] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek R. Narasayya. 2021. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *Proc. VLDB Endow.* 14, 13 (2021), 3376–3388.
- [14] Matthijs Douze, Alexander Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvassy, Pierre-Emmanuel Mazáré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024).
- [15] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *EMNLP*. 6894–6910.
- [16] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.
- [17] Goetz Graefe and David J. DeWitt. 1987. The EXODUS Optimizer Generator. In *SIGMOD*. 160–172.
- [18] Goetz Graefe and William J. McKenna. 1993. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *ICDE*. 209–218.
- [19] Yue Han, Guoliang Li, Haitao Yuan, and Ji Sun. 2021. An Autonomous Materialized View Management System with Deep Reinforcement Learning. In *ICDE*. 2159–2164.
- [20] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yeqin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12 (2023), 248:1–248:38.
- [21] Viktor Leis, Andrej Gubichev, Atanas Mirchev, Peter A. Boncz, Alfonso Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9 (2015), 204–215.
- [22] Feifei Li. 2019. Cloud native database systems at Alibaba: Opportunities and Challenges. *Proc. VLDB Endow.* 12, 12 (2019), 2263–2272.
- [23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as a Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. In *NeurIPS*.
- [24] Xiaonan Li, Kai Lv, Hang Yan, Tianyang Lin, Wei Zhu, Yuan Ni, Guotong Xie, Xiaoling Wang, and Xipeng Qiu. 2023. Unified Demonstration Retriever for In-Context Learning. In *ACL*. 4644–4668.
- [25] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL*. 142–150.
- [26] Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, Pengyuan Li, Kun-Lung Wu, Syed Zawad, Andrew Coleman, Matthew White, Mark Lewis, Raju Pavuluri, Yan Koyfman, Boris Lublinsky, Maximiliel de Bayser, Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Yi Zhou, Chris Johnson, Aanchal Goyal, Hima Patel, S. Yousaf Shah, Petros Zerfos, Heiko Ludwig, Asim Munawar, Maxwell Crouse, Pavan Kapanipathi, Shweta Salaria, Bob Caffo, Sophia Wen, Seetharami Seelam, Brian Belgodere, Carlos A. Fonseca, Amit Singhee, Nirmit Desai, David D. Cox, Ruchir Puri, and Rameswar Panda. 2024. Granite Code Models: A Family of Open Foundation Models for Code Intelligence. *CoRR* abs/2405.04324 (2024).
- [27] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. 1992. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *SIGMOD*. 39–48.
- [28] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*. 3980–3990.
- [29] Timo Schick, Jane Dwivedi-Yu, Roberto Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*.
- [30] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I Learned to Start Worrying about Prompt Formatting. *CoRR* abs/2310.11324 (2023).
- [31] Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. SQL-PALM: Improved Large Language Model Adaptation for Text-to-SQL. *CoRR* abs/2306.00739 (2023).
- [32] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lamplé. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).
- [33] Can Wang, Sheng Jin, Yingda Guan, Wentao Liu, Chen Qian, Ping Luo, and Wanli Ouyang. 2022. Pseudo-Labeled Auto-Curriculum Learning for Semi-Supervised Keypoint Localization. In *ICLR*.
- [34] Zhaoguo Wang, Zhou Zhou, Yicun Yang, Haoran Ding, Gansen Hu, Ding Ding, Chuzhe Tang, Haibo Chen, and Jinyang Li. 2022. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. In *SIGMOD*. ACM, 94–107.
- [35] Jerry W. Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Weisbo, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. 2023. Larger language models do in-context learning differently. *CoRR* abs/2303.03846 (2023).
- [36] Wentao Wu, Philip A. Bernstein, Alex Raizman, and Christina Pavlopoulou. 2022. Factor Windows: Cost-based Query Rewriting for Optimizing Correlated Window Aggregates. In *ICDE*. 2722–2734.
- [37] Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, Wang Zhao, Fan Zhou, Danrui Qi, Hong Yi, Shaodong Liu, and Faqiang Chen. 2023. DB-GPT: Empowering Database Interactions with Private Large Language Models. *CoRR* abs/2312.17449 (2023).
- [38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*.
- [39] Seonghyeon Ye, Jiseon Kim, and Alice Oh. 2021. Efficient Contrastive Learning via Novel Data Augmentation and Curriculum Learning. In *EMNLP*. 1832–1838.
- [40] Haitao Yuan, Guoliang Li, Ling Feng, Ji Sun, and Yue Han. 2020. Automatic View Generation with Deep Learning and Reinforcement Learning. In *ICDE*. 1501–1512.
- [41] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *CoRR* abs/2309.01219 (2023).
- [42] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. 2022. QueryFormer: A Tree Transformer Model for Query Plan Representation. *Proc. VLDB Endow.* 15, 8 (2022), 1658–1670.
- [43] Yue Zhao, Zhaodonghui Li, and Gao Cong. 2023. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. *Proc. VLDB Endow.* 17, 4 (2023), 823–835.
- [44] Yue Zhao, Zhaodonghui Li, and Gao Cong. 2024. A Comparative Study and Component Analysis of Query Plan Representation Techniques in ML4DB Studies. *Proc. VLDB Endow.* 17, 4 (2024), 823–835.
- [45] Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. 2021. A Learned Query Rewrite System using Monte Carlo Tree Search. *Proc. VLDB Endow.* 15, 1 (2021), 46–58.
- [46] Yuhang Zhou, He Yu, Siyu Tian, Dan Chen, Liuzhi Zhou, Xinlin Yu, Chuanjun Ji, Sen Liu, Guangnan Ye, and Hongfeng Chai. 2023. R³-NL2GQL: A Hybrid Models Approach for Accuracy Enhancing and Hallucinations Mitigation. *CoRR* abs/2311.01862 (2023).

참조

- [1] [n.d.]. Apache Calcite 재작성 규칙. <https://calcite.apache.org/javadocAggregate/org/apache/calcite/rel/rules/package-summary.html>. [2] [n.d.]. OpenAI 텍스트 생성 API 소개, <https://platform.openai.com/docs/guides/text-generation>. [3] [n.d.]. LLM 테이터베이스 관리자로서. <https://github.com/TsinghuaDatabaseGroup/DB-GPT>. [4] [n.d.]. PostgreSQL. <https://www.postgresql.org>. [5] [n.d.]. TPC-H 도구 모음. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp. [6] 아리안 앤스카리, 크리스티안 포엘리츠, 및 신예 탕. 2024. MAGIC: 인컨텍스트 텍스트-투-SQL을 위한 자기 수정 가이드라인 생성. CoRR abs/2406.12692 (2024). [7] 추시 바이, 사팀 알수다이스, 및 천 리. 2023. QueryBooster: 인간 중심 쿼리 재작성을 위한 미들웨어 서비스 사용 SQL 성능 향상. Proc. VLDB Endow. 16, 11 (2023), 2911–2924. [8] 에드먼 베글리, 헤수스 카마초-로드리고스, 줄리안 하이드, 마이클 J. 미오르, 및 다니엘 레미르. 2018. Apache Calcite: 이질적 데이터 소스 전반에 걸쳐 최적화된 쿼리 처리를 위한 초기 프레임워크. SIGMOD. 221–230. [9] 교수아 뱅시오, 제롬 루라도우르, 로랑 콜로베르, 및 제이슨 웨스트은. 2009. 교육 과정 학습. ICLM. Vol. 382. 41–48. [10] 톰 B. 브라운, 벤자민 만, 닉 라이더, 멜라니 서비아, 제드 카풀란, 프라풀라 다리왈, 아르빈드 네엘라칸탄, 프라나브 시암, 기리쉬 사스트리, 아만다 아셀, 산디니 아카왈, 아리엘 허버트-보스, 그雷첸 크루거, 톰 해니건, 리원 차일드, 아디티야 라메시, 다니엘 M. 지글러, 제프리 우, 클레멘스 윈터, 크리스토퍼 헤세, 마크 전, 에릭 시글러, 마테우스 리트원, 스콧 그레이, 벤자민 챕스, 캐스 클라크, 크리스토퍼 베너, 샘 맥캔들리시, 알렉 래드포드, 일리야 수트스케버, 및 다리오 아모데오. 2020. NeurIPS. [11] 파울라 카산데-보니아, 푸엘 탄, 얀준 치, 및 비센테 오르도네즈. 2021. 교육 과정 라밸링: 준지도 학습을 위한 가짜 라밸링 재방문. AAAI. 6912–6920. [12] 다니엘 M. 세르, 모나 T. 디아브, 에네코 아카리에, 이니고 로페즈-가스피오, 및 루시아 스피시아. [n.d.]. SemEval-2017 작업 1: 다국어 및 횡단어적 초점을 맞춘 의미론적 텍스트 유사성 평가. ACL. 1–14. [13] 배일루 당, 수라지트 차우두리, 요하네스 게르케, 및 비베른 R. 나라사야. 2021. DBS: 워크로드 구동 및 전통 데이터베이스 시스템을 위한 의사 결정 지원 벤치마크. Proc. VLDB Endow. 14, 13 (2021), 3376–3388. [14] 마티이스 두즈, 알렉산드르 구즈바, 당 청기, 제프 존슨, 게르겔리 실바시, 피에르-에마뉘엘 마자레, 마리아 로멜리, 루카스 호세이니, 및 에르베 제구. 2024. Faiss 라이브러리. CoRR abs/2401.08281 (2024). [15] 천유 가오, 싱청 야오, 및 단치 천. 2021. SimCSE: 간단한 대조 학습을 통한 문장 임베딩. EMNLP. 6894–6910. [16] 고츠 그라페. 1995. 카스케이드 프레임워크: 쿼리 최적화. IEEE Data Eng. Bull. 18, 3 (1995), 19–29. [17] 고츠 그라페 및 데이비드 J. 드윗. 1987. EXODUS 최적화 생성기. SIGMOD. 160–172. [18] 고츠 그라페 및 윌리엄 J. 맥켄나. 1993. 화산 최적화 생성기: 확장성 및 효율적인 검색. ICDE. 209–218. [19] 월한, 과리양 리, 하이타오 위안, 및 지 선. 2021. 심층 강화 학습을 이용한 자율적 물질학습 관리 시스템. ICDE. 2159–2164. [20] 자웨이 지, 나연 리, 리타 프리스케, 타이정 유, 단 수, 앙쉬, 에츠코 이시이, 예진 방, 안드레아 마도토, 및 패스칼레 평. 2023. 자연어 생성에 서 환각에 대한 조사. ACM Comput. Surv. 55, 12 (2023), 248:1–248:38. [21] 비크토르 라이프, 안드레이 구비체프, 아타나스 미르체프, 피터 A. 본츠, 알폰스 캠페, 및 토마스 노이만. 2015. 쿼리 최적화기의 실제 성능은 어떤가? Proc. VLDB Endow. 9 (2015), 204–215. [22] 페파이 리. 2019. 알리바바에서의 클라우드 네이티브 데이터베이스 시스템: 기회와 도전. Proc. VLDB Endow. 12, 12 (2019), 2263–2272. [23] 진양 리, 빈위안 후이, 게 쿼, 자시 양, 빈후아 리, 보웬 리, 백일린 왕, 보웬 친, 러이잉 경, 난 후오, 쉬안혜 주, 천하오 마, 과리양 리, 케빈 천-추안 장, 페이 황, 레이월드 청, 및 용빈 리. 2023. LLM 이미 데이터베이스 인터페이스로 기능할 수 있는가? 대규모 데이터베이스 그라운드 텍스트-투-SQL을 위한 BIG 벤치. NeurIPS. [24] 시아오난 리, 카이 루, 향 안, 티양 린, 웨이 주, 위안 니, 구통 세, 시아oling 왕, 및 시평 추. 2023. 인컨텍스트 러닝을 위한 통합 데몬스트레이션 리트리버. ACL. 4644–4668. [25] 앤드류 L. 마스, 레이몬드 E. 데일리, 피터 T. 팜, 단 황, 앤드류 Y. 응, 및 크리스토퍼 포츠. 2011. 감정 분석을 위한 단어 벡터 학습. ACL. 142–150. [26] 마얀크 미쉬라, 맷 스탈룬, 가오위안 장, 이강 선, 아디티아 프라사드, 아드리아나 메자 소리아, 미셸 메를리, 파마스와르네 셀바무, 사프타 수렌드라난.

시브딥 성, 마니쉬 세티, 쿠안-홍 당, 평위안 리, 쿤-룽 우, 사이드 자와드 앤드류 콜먼, 메튜 화이트, 마크 루이스, 라주 파볼리, 안 코이프만, 보리스 루블린스키, 막시밀리안 드 베이저, 이브라힘 암델아지즈, 킨잘 바수, 마얀크 아가왈, 이 저우, 크리스 존슨, 아안찰 고알, 히마 파텔, S. 유사프 사, 페트로스 제르포스, 하이코 루드비히, 아시무르 무나와르, 맥스웰 크루스, 파반 카파니파티, 수웨타 살라리아, 밥 칼리오, 소피아 웬, 시타라미 실람, 브라이언 벨고데레, 카를로스 A. 폰세카, 아미트 싱히, 널미트 데사이, 데이비드 D. 콕스, 루치르 푸리, 라메스와르 판다. 2024. 그래나이트 코드 모델: 코드 지능을 위한 오픈 소스 모델 가족. CoRR abs/2405.04324 (2024). [27] 하미드 파라헤시, 조셉 M. 헬러스타인, 와카르 하산. 1992. 확장 가능/규칙 기반 쿼리 재작성 최적화 스타벅스에서. SIGMOD. 39–48. [28] 닐스 라이머, 이리나 구레비치. 2019. 문장-BERT: 시아메스 BERT 네트워크를 사용한 문장 임베딩. EMNLP. 3980–3990. [29] 티모 슈리, 제인 드워디유, 로베르토 데시, 로베르타 레일레누, 마리아 로멜리, 에릭 햄브로, 루크 제틀모이어, 니콜라 칸체다, 토마스 시알룸. 2023. 투포머: 언어 모델은 도구 사용법을 스스로 가르칠 수 있다. NeurIPS. [30] 이 멜라니 스클라, 예진 죄, 유리아 즈베트코브, 알레인 수어. 2023. 언어 모델의 스퓨리어스 기능에 대한 민감도 정량화: 프롬프트 디자인 또는 프롬프트 포맷팅에 대해 걱정하기 시작한 방법. CoRR abs/2310.11324 (2023). [31] 루오시 선, 세르간 Ö. 아리, 후탄 나코스트, 칸준 다이, 라자리시 싱하, 평정 인, 토마스 포스터. 2023. SQL-PaLM: 텍스트-SQL을 위한 개선된 대형 언어 모델 적용. CoRR abs/2306.00739 (2023). [32] 허고 투브론, 티보 라브릴, 고티에 이자카드, 세비어 마르티네, 마리-안 라슈, 티모데 라크루아, 바티스트 로지에르, 나만 고얄, 에릭 햄브로, 피어살 아즈하르, 오렐리앙 로드리게즈, 아르망 주랑, 에드워드 그라브, 기요 람풀. 2023. LLaMA: 오픈 및 효율적인 재단 언어 모델. CoRR abs/2302.13971 (2023). [33] 왕간, 생 진, 잉다 판, 웬타오 류, 천 치안, 평 루오, 완리 오우양. 2022. 준지도 키포인트로 컬라 이제이션을 위한 가짜 레이블 자동 커리큘럼 학습. ICLR. [34] 자오구오 왕, 주주, 이춘 양, 하오란 당, 간센 후, 당 당, 추즈혜 당, 하이보 첸, 진양 리. 2022. WeTune: 쿼리 재작성 규칙의 자동 발견 및 검증. SIGMOD. ACM. 94–107. [35] 제리 W. 웨이, 제이슨 웨이, 이 테일, 더스틴 트랜, 알버트 웹슨, 이평 루, 신윤 챈, 한샤오 류, 다황, 데니 주, 텐유 마. 2023. 더 큰 언어 모델은 컨텍스트 학습을 다르게 한다. CoRR abs/2303.03846 (2023). [36] 웬타오 우, 필립 A. 베슈타인, 알렉스 라이즈만, 크리스티나 파블로풀루. 2022. 요인 창: 상관된 창 짍계 최적화를 위한 비용 기반 쿼리 재작성. ICDE. 2722–2734. [37] 시자오 쉐, 카이가오 지양, 웬후이 시, 팽인 첸, 케팅 첸, 흥준 양, 평핑 장, 지안산 해, 흥양 장, 강린 웨이, 왕 자오, 팬 주, 단루 치, 흥 이, 샤오동 류, 파강 첸. 2023. DB-GPT: 프라이빗 대형 언어 모델을 통한 데이터베이스 상호작용 강화. CoRR abs/2312.17449 (2023). [38] 순유 야오, 제프리 자오, 디안 유, 난 두, 이자크 사프린, 카르티 R. 나스림한, 원 카오. 2023. ReAct: 언어 모델에서 추론과 작용의 시너지. ICLR. [39] 성현 예, 지선 김, 앤리스 오. 2021. 효율적인 대조 학습을 위한 새로운 데이터 증강 및 커리큘럼 학습. EMNLP. 1832–1838. [40] 하이타오 위안, 과리양 리, 링 평, 지 선, 월한. 2020. 딥 러닝과 강화 러닝을 사용한 자동 뷰 생성. ICDE. 1501–1512. [41] 월장, 야푸 리, 레이양 추이, 당 차이, 레마오 류, 텅진 푸, 신팅 황, 앤보 자오, 유 장, 유통 천, 룽위 왕, 안투안 루, 웨이 비, 프레다 시, 슈밍 시. 2023. AI 해양의 시렌 노래: 대형 언어 모델에서 환각에 대한 조사. CoRR abs/2309.01219 (2023). [42] 월자오, 가오 콩, 지아첸 시, 출안 미아오. 2022. 쿼리포머: 쿼리 계획 표현을 위한 트리 트랜스포머 모델. Proc. VLDB Endow. 15, 8 (2022), 1658–1670. [43] 월자오, 자오동후이 리, 가오 콩. 2023. ML4DB 연구에서 쿼리 계획 표현 기술에 대한 비교 연구 및 구성 요소 분석. Proc. VLDB Endow. 17, 4 (2023), 823–835. [44] 월자오, 자오동후이 리, 가오 콩. 2024. ML4DB 연구에서 쿼리 계획 표현 기술에 대한 비교 연구 및 구성 요소 분석. Proc. VLDB Endow. 17, 4 (2024), 823–835. [45] 쇠안혜 주, 과리양 리, 청량 차이, 전화 평. 2021. 몬테카를로 트리 탐색을 사용한 학습 쿼리 재작성 시스템. Proc. VLDB Endow. 15, 1 (2021), 46–58. [46] 유향 주, 허 유, 시유 티안, 단 첸, 류지 주, 신린 유, 추안준 지, 센류, 광난 예, 홍풍 차이. 2023. R3-NL2GQL: 정확도 향상과 환각 완화를 위한 하이브리드 모델 접근 방식. CoRR abs/2311.01862 (2023).