Generating Cross-model Analytics Workloads Using LLMs

Xiuwen Zheng
xiz675@ucsd.edu
University of California, San Diego
La Jolla, USA

Arun Kumar
akk018@ucsd.edu
University of California, San Diego
La Jolla, USA

Amarnath Gupta
a1gupta@ucsd.edu
University of California, San Diego
La Jolla, USA

Abstract

Data analytics applications today often require processing heterogeneous data from different data models, including relational, graph, and text data, for more holistic analytics. While query optimization for single data models, especially relational data, has been studied for decades, there is surprisingly little work on query optimization for *cross-model* data analytics. Cross-model query optimization can benefit from the long line of prior work in query optimization in the relational realm, wherein cost-based and/or machine learning-based (ML-based) optimizers are common. Both approaches require a large and diverse set of *query workloads* to measure, tune, and evaluate a query optimizer. *To the best of our knowledge, there are still no large public cross-model benchmark workloads, a significant obstacle for systems researchers in this space.* In this paper, we take a step toward filling this research gap by generating new query workloads spanning relational and graph data, which are ubiquitous in analytics applications. Our approach leverages large language models (LLMs) via different prompting strategies to generate queries and proposes new rule-based post-processing methods to ensure query correctness. We evaluate the pros and cons of each strategy and perform an in-depth analysis by categorizing the syntactic and semantic errors of the generated queries. So far, we have produced over 4000 correct cross-model queries, the largest set ever. Our code, prompts, data, and query workloads will all be released publicly.

CCS Concepts

• **Information systems** → **Database management system engines**; **Graph-based database models**; **Database design and models**.

Keywords

Cross-data model applications, Cross-data model benchmarks, LLM-powered data augmentation

ACM Reference Format:
Xiuwen Zheng, Arun Kumar, and Amarnath Gupta. 2024. Generating Cross-model Analytics Workloads Using LLMs. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3627673.3679932

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679932

1 Introduction

Analytics over heterogeneous datasets spanning various data models (e.g., relational, graph, text) from different sources are increasingly common in fields like social sciences [14, 19] and cybersecurity [10–12]. Prior work shows that single data model stores may be inefficient for cross-model applications [16, 21]. To mitigate this, recent research on so-called *polystore systems* has introduced a middleware layer over different "unistore" DBMSs. This approach delegates parts of queries to suitable unistores, avoiding the need to transform and store different data types in a single store while leveraging each unistore's strengths.

Polystores support *cross-model queries* spanning data models, but optimizing these queries remains a research challenge. *The absence of a public large cross-model query benchmark hinders progress in polystore research.* Inspired by the impact of benchmarks like TPC-H on relational DBMSs, *this exploratory paper takes a step toward developing a cross-model benchmark for relational and graph models.*

1.1 Challenges

Generating large and diverse query workloads is crucial for evaluating polystore query optimizers, whether cost-based or ML-based. And specifically, ML-based optimizers require extensive training datasets. While this problem of query workload generation is already challenging for just RDBMSs, it is even more compounded for cross-model applications due to additional data models. To the best of our knowledge, there is no public large query workload for cross relational-graph analytics. A possible solution is to adapt relational benchmarks like TPC-H to create a cross-model benchmark, but these lack the complex relationships among entity tables needed for graph-oriented analytics.

Thus, we modify a popular graph benchmark, LDBC[1], which includes complex relationships among node entities and multiple node properties for each of the 14 entities. Figure 1 shows the original graph schema with hidden node properties for brevity. We place relationship data in the graph store and node property data in relational tables to create a cross-model schema shown in Figure 2.

To produce cross-model queries, we could modify LDBC graph queries (Cypher) to generate a mix of Cypher and SQL. However, the LDBC benchmark has only 22 queries. Worse, not all suitable for cross-model analytics as they do not need relational-style capabilities. A lot more cross-model queries need to be generated. The queries should be diverse enough, spanning different levels of complexity on both the SQL side and Cypher side. So, we cannot just use a small template and modify its predicates, an approach taken by prior SQL-generating methodologies such as Lero [22].

[1] https://ldbcouncil.org/benchmarks/snb-bi/

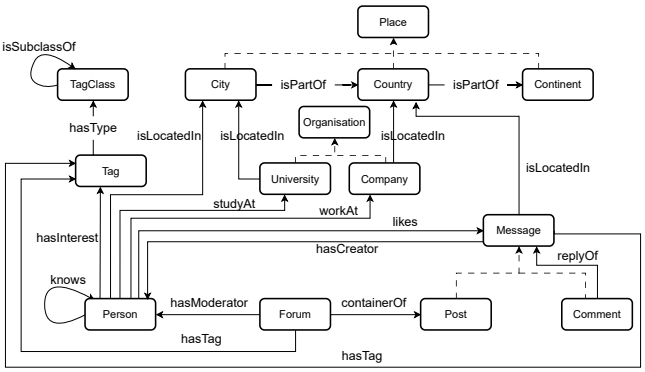

Figure 1: LDBC Graph Dataset.

Graph Schema

Nodes property:	Relationships:
Country: id	Place-[IS_PART_OF]->Place
City: id	TagClass-[IS_SUBCLASS_OF]->TagClass
Place: id	Organisation-[IS_LOCATED_IN]->Place
Organisation: id	Tag-[HAS_TYPE]->TagClass
University: id	Comment-[HAS_CREATOR]->Person
Company: id	Comment-[IS_LOCATED_IN]->Country
TagClass: id	Comment-[REPLY_OF]->Comment
Tag: id	Comment-[REPLY_OF]->Post
Forum: id	Forum-[CONTAINER_OF]->Post
Person: id	Forum-[HAS_MEMBER]->Person
Post: id	Forum-[HAS_MODERATOR]->Person
Comment: id	Forum-[HAS_TAG]->Tag
Message: id	.....

Relation Schema

Country <id, name>
City <id, name>
Place <id, name>
Organisation <id, name>
University <id, name>
Company <id, name>
TagClass <id, name>
Tag <id, name>
Forum <id, title, creationDate>
Person <id, firstName, lastName, gender, ...>
Post <id, creationDate, content, length>
Comment <id, creationDate, content, length>
Message <id, creationDate, content, length>

Figure 2: Schema of Modified Data.

1.2 Our Contributions

In this exploratory paper, we address the above challenges by leveraging recent large language models (LLMs) that have been shown to have remarkable text-to-SQL translation performance [3, 6, 15] and potential in data augmentation [13, 18, 20]. We explore how to prompt LLMs to develop and standardize a cross-model query benchmark spanning SQL and Cypher queries. We explore *three different prompting strategies presenting the database schemas differently to the LLM.* Table 1 illustrates these strategies, and Section 2 explains them in more detail. We find the strategy utilizing a virtual graph view for cross-model data works best. We *propose a mechanism to automatically categorize, verify, and correct some errors* in LLM-generated queries and a *post-processing procedure to automatically rewrite generated Cypher queries into cross-model queries.* Using our approach, we produced over 4000 correct queries, the largest set for such cross-model applications. All our code, prompts, data, and generated queries will be made public on GitHub to spur further research and optimize polystore systems.

1.3 Related work

Polystore systems and workloads. Many existing polystore systems lack a large set of workloads for development or evaluation. Myria [17] and CloudMdsQL [7] evaluate on two multi-model analytical workloads. BigDAWG [4, 5] and RHEEM [1, 8] evaluate on three workloads. ESTOCADA [2] evaluates on 50 queries generated from 25 templates, the largest set of cross-model workloads to date. However, ESTOCADA focuses on relational, JSON, and textual data, lacking graph data or queries. Currently, no large cross relational-graph analytics benchmark is available.

SQL Query Workload Augmentation. While there are some benchmarks for SQL, they are small, e.g., JOB [9] has 100 queries, and TPC-H has 22. To expand the query set, methods like Lero [22] generate 1000 queries from these benchmarks by adding or modifying predicates. However, these generated queries often lack diversity and may result in identical query plans, reducing their utility.

LLMs for data augmentation. There are several studies on text-to-SQL generation using LLMs [3, 6, 15]. In other domains, LLMs have been utilized for data augmentation for tasks like Chinese dialogue–level dependency parsing [20], multilingual commonsense reasoning [18] and event extraction in natural language processing [13], where available data is limited. However, to the best of our knowledge, there are no existing studies using LLMs to generate or augment a large set of queries for even single data model.

2 Strategies for query generation

For tractability purposes, in this initial work we focus on one class of cross-model queries: one Cypher query followed by an SQL query using the Cypher result as a table. We explore 3 different ways to use LLMs for generating queries by altering how database schema is given in the prompt. Table 1 lists the strategies; Figure 3 lists their corresponding prompts.

Strategy 1 (S1) uses the actual graph and relation schema (Figure 2), and asks the LLM to generate random cross-model queries. Figure 4 (a) shows an example output with a Cypher and an SQL query. The advantage is that the generated queries are cross-model queries that can be directly used on the underlying stores. However, the output tokens have redundant information. For example, the node property `city.id` from Cypher RETURN clause and the `cityId` column of relation *ge* in the SQL SELECT clause refer to the same property and appear twice. Additionally, SQL queries often include many tokens related to joins. This can be costly if a large number of queries are generated, as token-based pricing is common for LLM APIs. Moreover, it is not straightforward to automatically verify the correctness of the generated queries, as both queries on different stores need to be verified. The SQL query uses the result from the Cypher query, so the relational store's planner cannot evaluate the SQL query without first running the Cypher query and storing the result as a relation in relational store.

S2 uses a *virtual graph view* and asks the LLM to generate purely Cypher queries. In the virtual graph schema, all relation columns are expressed as the corresponding node's properties. The output is much more concise than that of the S1, as shown in Figure 4 (b), since it hides the joins between nodes and relations, and extracted properties only appear once in the RETURN clause. We conducted an initial experiment to generate 100 queries using Strategies 1 and 2, and the average number of tokens for these queries were 42 and 91, respectively. It is also easier to evaluate the correctness of the generated queries, as only one type of query needs to be checked. We can use graph store planners, such as Neo4j, to evaluate the query without running it. However, a post-processing procedure is needed to rewrite each pure Cypher query into a cross-model query with a Cypher query followed by an SQL query.

S3 uses an integrated *virtual relational view* and asks the LLM to generate purely SQL queries. The graph can be expressed by several relations, with node properties added as columns to the

Table 1: Comparison of Different Strategies

	Database schema	Pros	Cons
Strategy 1 (S1)	Both data models	No query rewriting needed	More expensive due to longer token lengths Verification is more complex
Strategy 2 (S2)	Virtual graph schema	# of output tokens is about half Verification is easy	Simple and deterministic post-processing
Strategy 3 (S3)	Virtual relation schema	Same as Strategy 2	Post-processing is more complex than Strategy 2

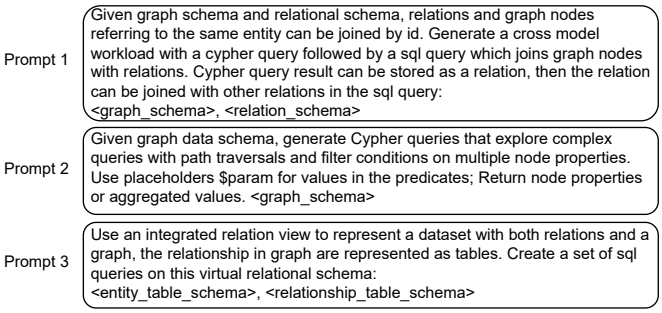

Figure 3: Prompts of Different Strategies.

```
MATCH (person:Person)-[r:WORK_AT|STUDY_AT]->(org:Organisation),
(org)-[:IS_LOCATED_IN]->(city:City)
RETURN person.id AS personId, org.id AS orgId, org.name AS orgName,
city.id AS cityId, city.name AS cityName, TYPE(r) as relationType;

SELECT ge.personId, p.firstName, p.lastName, p.email, ge.orgId, ge.orgName,
ge.cityId, c.name AS cityName, ge.relationType
FROM graph_employment ge JOIN Person p ON ge.personId = p.id JOIN
Organisation o ON ge.orgId = o.id JOIN City c ON ge.cityId = c.id;
```

(a) Example output from Strategy 1

```
MATCH (p:Person)-[:LIKES]->(comment:Comment)-[:HAS_TAG]->(tag:Tag)
MATCH (p)-[:IS_LOCATED_IN]->(city:City)
WHERE tag.name = $param AND city.name = $param
RETURN p.firstName, p.lastName, comment.content;
```

(b) Example output from Strategy 2

Figure 4: Example Output from Different Strategies.

corresponding entity tables, and each type of relationship can be viewed as a table with start node IDs and end node IDs as two columns. Like S2, it saves cost by producing more concise outputs but requires a post-processing step.

Given the many cons of S1, we use a virtual integrated view in the prompt. Between Strategies 2 and 3, we pick the former due to two reasons. First, the post-processing for S3 is much more complex–it needs to understand the entire logic of the query to extract the join graph between tables and then translate the join graph into paths on the graph data, which is hard to automate. Second, it is hard for LLMs to generate SQL queries that translate into certain path patterns. For example, variable-length paths need to be expressed using recursive SQL queries, which are large and cumbersome. Unless explicitly specified in the prompt, it is not common to find this pattern in the generated SQL queries.

Figure 5 shows the whole workflow of using Strategy 2. We use the LLM to generate a set of Cypher queries on the virtual graph view. Then the queries will be verified or corrected (Section 3). Each correct query will be rewritten into a Cypher query + an SQL query on the real cross-model dataset (Section 4).

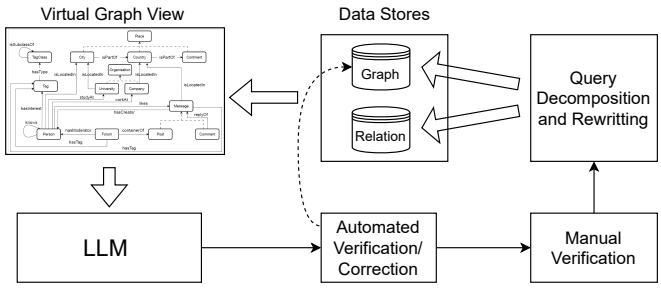

Figure 5: Workloads Generation Framework with Strategy 2.

3 Verification and Correction of LLM-generated queries

To verify the Cypher queries generated with Strategy 2, we transform and store all data, including relations, in Neo4j, a graph database, subject to the virtual graph view. We ask Neo4j to run `EXPLAIN` on each query; it generates an estimated plan and output cardinality without executing the query. Using that information, we categorize all queries into 6 main classes, as listed in Table 2.

If there is a syntax error when running `explain <query>`, the query is categorized as Category 2 (C2). Queries without syntax errors are further examined for semantic errors, which may result in no output being returned. For syntax-correct queries, we assess the estimated output cardinality. If the cardinality is relatively large, the query is likely semantically correct and is placed in Category 1 (C1). For queries with a small estimated output cardinality, we observed common semantic mistakes:

- Some edge/node types do not exist in the graph (C3).
- The edge type exists, but should not exist between the two nodes it connects (C4).
- The edge direction between two nodes is incorrect (C5).

Queries may fall into multiple categories among C3-C5 if they exhibit multiple types of semantic errors. If small estimated cardinality queries do not exhibit any of the above errors, they are placed in an unspecified category (C6). For C5, if the queries only have this type of semantic error, we wrote a program to correct the edge directions. After fixing the directions, we run `EXPLAIN` on the new query; if the new estimated cardinality is relatively large, the query is placed in subcategory C5.2. Otherwise, it is placed in C5.3. Queries in C5.3 are included in C6 for further observation. There are 406 queries in C5.2, meaning they are semantic correct after fixing the direction errors.

LLM-generated queries are automatically categorized by our program. Queries in C6 are then manually checked and put into several sub-categories. Some queries (C6.2) return an aggregated value, resulting in an estimated cardinality of 1, which is below

Table 2: Categories of LLMs-generated Queries

Descriptions of Categories	# Queries	Sub-categories
C1: Syntax and semantic correct	3695	
C2: Syntax error	12	C2.1: non existing functions; C2.2: not using specified separators for queries; C2.3: reuse variable name in one match clause; C2.4: others
C3: Non-existing edge/node type	6	
C4: Wrong edge type for specific nodes	37	
C5: Wrong edge directions	436	C5.1: also in C3 or C4, no correction for them C5.2: not in C3/C4; fall in C1 after fixing the directions (406) C5.3: not in C3/C4; have small estimated cardinality even after fixing
C6: Unspecified	234	C6.1: Inaccurate estimation of Neo4j's planner C6.2: Aggregation function such as count(*) returned (11) C6.3: Incompatible functions for some data types C6.4: Predicates are too selective

our threshold for semantic error-free queries, but these queries are semantically correct. All queries in C2, C5.2, and C6.2 are selected as the final set of correct queries. Out of the 4,390 generated queries, a total of 4,112 ($\sim$ 94%) are correct.

4 Post-processing: Query Decomposition and Rewriting

We apply a post-processing procedure to rewrite the correct Cypher queries on the virtual view to queries on the real schema. For brevity, we skip the detailed algorithm and provide a succinct summary. Our algorithm takes a Cypher query Q and a map that stores the properties that are in tables but not in the graph for each node type.

Q is parsed into individual clauses; each is processed differently and helps form different parts of the SQL query:

- **MATCH or WHERE Clause:** Labels for variables in MATCH clause are identified. For each predicate, if the variable's property is not in graph, then the predicate is removed from this clause and added to the SQL WHERE clause instead. The variable is added to a join list ($joinVarList$) that keeps track of variables needed to be joined with tables in SQL queries.
- **WITH Clause:** All variables in $joinVarList$ are added there.
- **RETURN Clause:** Each returned value is added to the SELECT clause of the SQL query. If a returned value uses a property not in graph, the return clause is changed to return ID and the variable is added to $joinVarList$. If a returned value is an aggregation function on a variable, the return clause is also changed to return its ID, and all other returned values in the original RETURN clause are added to the SQL GROUP BY clause. For variables in $joinVarList$, their IDs are added to the RETURN clause, and they are joined with corresponding tables on ID in the SQL WHERE clause and these tables are added to SQFROM clause.

5 Analysis on the cross-model queries

For the LLM, we evaluated Llama3 and OpenAI's GPT-4o. We found that GPT-4o generates the most error-free queries with varying complexities. Cypher queries typically match a path pattern from the graph, and we use path length as an evaluation metric of complexity. For SQL queries, we use the number of tables involved as a metric. Each model was used to generate 100 queries. Llama3 had a correct rate of only 73%, while GPT-4o had a rate of 95%. We rewrote correct queries to Cypher and SQL queries on the real schema. For Llama3, 79% of Cypher queries matched a path with a length of 2, and 100% of the SQL queries involved 2 or 3 tables. GPT-4o performed better in terms of variety: the lengths of fixed-length paths ranged from 0 to 6 and the number of tables in SQL queries ranged from 0 to 5.

Table 3: Simple Statistics of Cross-model Queries

(a) Path Length Distribution of Cypher Queries

Path length	0	1	2	3	4	5	6	Variable
# queries	6	77	1926	1131	113	7	3	846

(b) Number of Tables Selected from of SQL Queries

# tables	0	1	2	3	4	5	6
# queries	4	210	1061	2107	679	47	1

Thus we used GPT-4o to generate more queries. In each request, we ask it to generate 10 Cypher queries, continuing the requests until we reach our budget limit ($6). After verification and correction, 4,112 out of 4,390 generated queries were correct; then a post-processing procedure was applied to obtain cross-model queries. Table 3 shows some summary statistics on the complexity of the cross-model queries.

Three queries with corner cases can not be processed by our current algorithm. Among the remaining queries, 6 Cypher queries had empty paths and 4 SQL queries did not involve any table. These 10 queries are not cross-model queries as they only query one data model. The remaining 4,099 queries are cross-model queries.

6 Conclusion and Ongoing Work

In this exploratory paper, we take a step toward generating a new large cross-model query benchmark spanning relational and graph data by leveraging state-of-the-art LLMs. In ongoing work, we are designing a query diversity metric to ensure the benchmark stresses different aspects of the underlying data systems, such as graph algorithms in the graph query. We will also use our benchmark to study new cross-model query optimization methods, including new deep learning-based approaches that were hitherto impossible without a large query workload like this.

Acknowledgments

This work was supported in part by NSF under award numbers 1909875 and 1942724. The content is solely the responsibility of the authors and does not necessarily represent the views of NSF.

References

[1] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, et al. 2018. Rheem: enabling cross-platform data processing: may the big data be with you! *Proceedings of the VLDB Endowment* 11, 11 (2018), 1414–1427.

[2] Rana Alotaibi, Bogdan Cautis, Alin Deutsch, Moustafa Latrache, Ioana Manolescu, and Yifei Yang. 2020. ESTOCADA: towards scalable polystore systems. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2949–2952.

[3] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306* (2023).

[4] Jennie Duggan, Aaron Elmore, Tim Kraska, Sam Madden, Tim Mattson, and Michael Stonebraker. 2015. The bigdawg architecture and reference implementation. *New England Database Day* (2015).

[5] Aaron Elmore, Jennie Duggan, Michael Stonebraker, Magdalena Balazinska, Ugur Cetintemel, Vijay Gadepally, Jeffrey Heer, Bill Howe, Jeremy Kepner, Tim Kraska, Samuel Madden, David Maier, Timothy Mattson, Stavros Papadopoulos, Jeff Parkhurst, Nesime Tatbul, Manasi Vartak, and Stan Zdonik. 2015. A Demonstration of the BigDAWG Polystore System. *Proc. Very Large Database Endowment (PVLDB)* 8, 12 (2015). http://idl.cs.washington.edu/papers/bigdawg-demo

[6] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[7] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. 2016. The cloudmdsql multistore system. In *Proceedings of the 2016 International Conference on Management of Data*. 2113–2116.

[8] Sebastian Kruse, Zoi Kaoudi, Bertty Contreras-Rojas, Sanjay Chawla, Felix Naumann, and Jorge-Arnulfo Quiané-Ruiz. 2020. RHEEMix in the data jungle: a cost-based optimizer for cross-platform systems. *The VLDB Journal* 29 (2020), 1287–1310.

[9] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215.

[10] Yang Lu and Li Da Xu. 2018. Internet of Things (IoT) cybersecurity research: A review of current research topics. *IEEE Internet of Things Journal* 6, 2 (2018), 2103–2115.

[11] Joash Mageto. 2021. Big data analytics in sustainable supply chain management: A focus on manufacturing supply chains. *Sustainability* 13, 13 (2021), 7101.

[12] Nishita Mehta and Anil Pandit. 2018. Concurrence of big data analytics and healthcare: A systematic review. *International journal of medical informatics* 114 (2018), 57–65.

[13] Zihao Meng, Tao Liu, Heng Zhang, Kai Feng, and Peng Zhao. 2024. CEAN: Contrastive Event Aggregation Network with LLM-based Augmentation for Event Extraction. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. 321–333.

[14] Vaia Moustaka, Athena Vakali, and Leonidas G Anthopoulos. 2018. A systematic review for smart city data analytics. *ACM Computing Surveys (cSuR)* 51, 5 (2018), 1–41.

[15] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2024).

[16] Michael Stonebraker and Uğur Çetintemel. 2018. " One size fits all" an idea whose time has come and gone. In *Making databases work: the pragmatic wisdom of Michael Stonebraker*. 441–462.

[17] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, et al. 2017. The Myria Big Data Management and Analytics System and Cloud Services.. In *CIDR*.

[18] Chenxi Whitehouse, Monojit Choudhury, and Alham Fikri Aji. 2023. Llm-powered data augmentation for enhanced crosslingual performance. *arXiv preprint arXiv:2305.14288* (2023).

[19] Christian Wolff and Thomas Schmidt. 2021. *Information between Data and Knowledge: Information Science and its Neighbors from Data Science to Digital Humanities*. Vol. 74. Werner Hülsbusch.

[20] Meishan Zhang, Gongyao Jiang, Shuang Liu, Jing Chen, and Min Zhang. 2024. LLM–Assisted Data Augmentation for Chinese Dialogue–Level Dependency Parsing. *Computational Linguistics* (2024), 1–24.

[21] Xiuwen Zheng, Subhasis Dasgupta, Arun Kumar, and Amarnath Gupta. 2023. An Optimized Tri-store System for Multi-model Data Analytics. *CoRR* abs/2305.14391 (2023).

[22] Rong Zhu, Wei Chen, Bolin Ding, Xingguang Chen, Andreas Pfadler, Ziniu Wu, and Jingren Zhou. 2023. Lero: A learning-to-rank query optimizer. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1466–1479.