Text to Graph Query Using Filter Condition Attributes

Yang Liu
Tianjin University
Tianjin, China
liruxru@tju.edu.cn

Xin Wang
Tianjin University
Tianjin, China
wangx@tju.edu.cn

Jiake Ge
Tianjin University
Tianjin, China
gejiake@tju.edu.cn

Hui Wang
Tianjin University
Tianjin, China
wanghui025@tju.edu.cn

Dawei Xu
Tianda Zhitu Co. Ltd.
Tianjin, China
xudawei@techfantasy.cn

Yongzhe Jia*
Tianjin University
Tianda Zhitu Co. Ltd.
Tianjin, China
jiayongzhe@tju.edu.cn

ABSTRACT

Converting natural language (NL) queries into formal graph query languages (GQL) is complex and requires specialized knowledge. Conventional methods are constrained by non-standardized datasets and limited node type diversity, hindering effective graph query generation. Moreover, utilizing database schema as contextual information seldom achieves high precision in identifying critical entity attributes. In order to address these limitations, this paper introduces a novel approach, that is termed Filter Condition Attribute Values (FCAV), which aims to augment the generation of Text-to-GQL. The initial datasets are constructed using template-filling and problem rewriting by large language models (LLMs). By integrating FCAV from the database into the input, our method enriches contextual information available to LLMs, enhancing the ability to discern the intricate relationships among NL query, graph schema, and actual data in the database. The approach is validated using two purpose-built datasets: TCMGQL (Traditional Chinese Medicine) and EduGQL (education). The experimental results show that our method outperforms traditional approaches on both non-fine-tuned and fine-tuned LLMs, and achieves $2 \sim 3\%$ better performance in domain migration tests.

VLDB Workshop Reference Format:
Yang Liu, Xin Wang, Jiake Ge, Hui Wang, Dawei Xu, and Yongzhe Jia. Text to Graph Query Using Filter Condition Attributes. VLDB 2024 Workshop: 3rd International Workshop on Large-Scale Graph Data Analytics (LSGDA 2024).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/liruxru/FCAV-train.

1 INTRODUCTION

Graph databases (GraphDBs), such as Neo4j [20], NebulaGraph [29], and Amazon Neptune [2] have become essential tools due to their efficiency and scalability in managing complex relational data. Processing natural language (NL) queries enhances user interaction with databases by eliminating the necessity for users to master complex query languages. This accessibility benefits non-technical users, broadening the applicability of data queries across various domains including knowledge-based question answering systems [6], voice assistants [3], and other natural language processing applications.

There are several challenges associated with transforming NL to graph query languages (NL2GQL). Firstly, the inherent ambiguity of NL can cause a single sentence to convey different meanings in different contexts, requiring deep linguistic comprehension from the model to parse user intent accurately [15]. Secondly, NL2GQL involves detailed grammatical and semantic analysis, necessitating not only keyword identification but also an understanding of sentence structure and context to generate correct query statements. Thirdly, the unique syntax and semantics of GQL, tailored to the different structures of GraphDBs, require the conversion system to understand not only NL, but also the query characteristics and data patterns specific to GraphDBs [8].

The advancement of large language models (LLMs) have led to a critical area of research: the effective integration of these models with GraphDBs to convert NL queries into formal GQL. While these studies[16, 18, 19] have produced some promising outcomes. However, existing datasets, such as SpCQL [10], demonstrate limited diversity and a lack of standardization. The majority of datasets use only two labels, “node" and “relationship", which imposes constraints on model training and negatively impacts on the generalization capability and practical utility of models. Liang [17] and Zhong [32] have both achieved significant success by fine-tuning LLMs using schema information binding with GraphDBs. However, the mapping process is often obscured by the ambiguity of NL and the abstraction of database schema. For example, using a specific attribute value from the database as a query keyword and relying solely on schema information can lead to GQL statements that inaccurately reflect the essential entity information. This inaccuracy arises because the schema does not specify the corresponding label or attribute key for the given attribute.

In order to bridge these gaps, we construct the TCMGQL and EduGQL datasets from real-world databases, ensuring standardized type and diversity. We develop over ten NL and GQL templates based on database schema information, enhanced by LLMs. The

*Corresponding author.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

proposed Filter Condition Attribute Values (FCAV) method uses database schema and attribute values as inputs to fine-tune LLMs, improving GQL generation accuracy. Experimental evaluations on various LLMs, confirms the effectiveness of our approach.

In this paper, our contributions can be summarized as follows:

(1) The FCAV is introduced as a comprehensive pipeline where attribute values of real data are integrated with GQL generation, leveraging the strengths of LLMs.

(2) A foundational paradigm for the creation of training datasets is presented.

(3) Experimental evaluations demonstrate that this approach yields significantly more accurate results than baseline methods.

We introduce the related work in Section 2. Section 3 outlines the preliminaries, while Section 4 details our pipeline. Section 5 presents the dataset, and Section 6 analyzes the experimental results. Section 7 discusses the limitations of our study. Finally, section 8 draws a conclusion.

2 RELATED WORK

2.1 Traditional Approaches to NL2GQL

Historically, NL2GQL have relied heavily on manual rule formulation [5, 31]. GraphAware NLP, a plugin for Neo4j developed by Albertodelazzari [4], represents a significant advancement, offering a comprehensive suite of tools. Similarly, the MANTRA QA system by Oro [23] designed to convert NL queries into SPARQL and Cypher statements. However, the disadvantages of these approaches are domain dependence on experts and involvement of labor in the process of development. Recent studies prompt a shift towards data-driven approaches. Many researchers have adopted Seq2Seq models [24, 25, 33] and hybrid techniques, integrating diverse models to enhance performance. However, the way databases are modeled depends heavily on domain-specific knowledge, making it difficult to generalize these approaches across domains.

2.2 NL2GQL with LLM

LLMs are advanced artificial intelligence models trained on extensive text data to accurately understand and generate human-like language [9]. The integration of LLMs with database systems have revolutionized the processing of NL to structured queries, due to the extensive coverage, generalization capabilities, and profound NL understanding of LLMs [7, 12, 21]. Despite their advancements, LLMs face challenges in aligning with domain-specific database knowledge, crucial for knowledge-intensive tasks [13]. Recent researchers aim at bridging this gap, Zhong [32] presents the SyntheT2C dataset, which employs LLMs for generating synthetic Query-Cypher pairs. Kang [14] introduces the SURGE method, which leverages subgraph retrieval to improve text consistency. Furthermore, Liang [17] has demonstrated the potential for aligning LLMs with GraphDBs using a dataset binding with schema info of graph, database knowledge is integrated into LLMs through fine-tuning using low-rank adaptation (LoRA) [11], thereby fundamentally enhancing generation accuracy.

However, these methods often overlook the dynamic and intricate nature of real data, particularly when attribute values are used as query keywords, which are crucial for generating accurate GQL. For example, as illustrated in Figure 1, if the NL input lacks specific

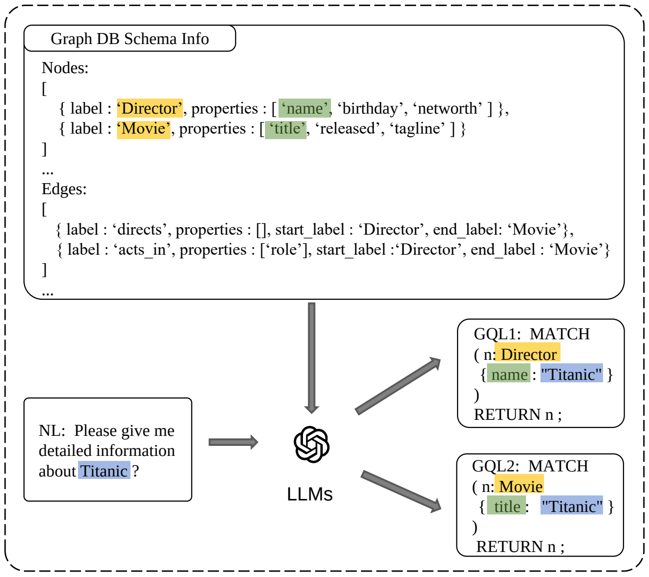

Graph DB Schema Info
Nodes:
[
{ label : 'Director', properties : ['name', 'birthday', 'networth'] },
{ label : 'Movie', properties : ['title', 'released', 'tagline'] }
]
...
Edges:
[
{ label : 'directs', properties : [], start_label : 'Director', end_label: 'Movie'},
{ label : 'acts_in', properties : ['role'], start_label :'Director', end_label : 'Movie'}
]
...
NL: Please give me detailed information about Titanic ?
LLMs
GQL1: MATCH (n: Director { name : "Titanic" }) RETURN n ;
GQL2: MATCH (n: Movie { title : "Titanic" }) RETURN n ;

Figure 1: Example of NL2GQL conversion using LLMs

keywords, such as "Movie", to indicate the label for "Titanic", a non-fine-tuned LLM might struggle to determine whether to use the "name" attribute key for a "Director" or the "title" attribute key for a "Movie" to generate GQL.

This paper bridges this gap with a dual strategy. First, we systematize graph query templates to fine-tune LLMs, allowing them to learn the relationships among NL, the graph schema, and the actual data in the graph. Second, we introduce the FCAV approach to enhance both the graph and the underlying LLMs, creating a comprehensive pipeline that aligns LLM knowledge with GraphDBs.

3 PRELIMINARIES

3.1 Property Graph

Definition 1 **(Property Graph Model).** A property graph G is defined as a 5-tuple $(V, E, \rho, \lambda, \sigma)$ where: V is a finite set of vertices; E is a finite set of edges, with $V \cap E = \emptyset$; $\rho : E \to (V \times V)$ is a function that associates each edge with a pair of vertices, for example, $\rho(e) = (v_1, v_2)$ denotes a directed edge e from vertex v_1 to v_2; $\lambda : (V \cup E) \to Lab$ is a labeling function mapping vertices and edges to sets of labels, such that for a vertex v (or an edge e), $\lambda(v)$ (or $\lambda(e)$) denotes its label; $\sigma : (V \cup E) \times Prop \to Val$ maps a vertex or edge and a property from the set $Prop$ to a value in Val, such that for $v \in V$ (or $e \in E$), $p \in Prop$, the value of property p on vertex v (or edge e) is $\sigma(v, p) = \text{val}$.

Figure 2 illustrates a property graph of a movie knowledge graph. In this graph, each vertex and edge is assigned a unique ID, labels, and a set of properties comprising a property name and value.

3.2 Graph Query Language

GQL is an advanced query language designed for the retrieval and manipulation of data in GraphDBs. GQL queries primarily consist of the following components:

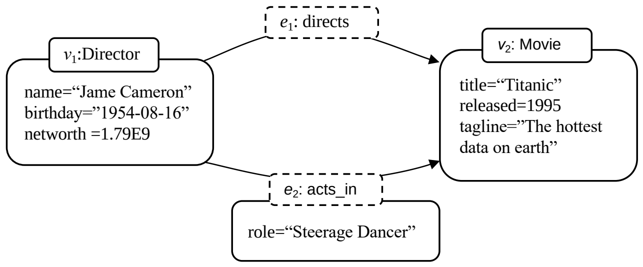


Figure 2: Example of a property graph

- **Pattern Matching.** Given a graph G and a query pattern $Q = (V_q, E_q)$, the objective of pattern matching is to find a subgraph $G' = (V', E')$ of G, such that G' is isomorphic to Q. This means there exists a bijection $f : V_q \to V'$ such that $(u, v) \in E_q \iff (f(u), f(v)) \in E'$.
- **Filter Conditions.** Filter conditions can be represented by predicate logic formulas. For example, for an attribute p of v and a value a, the filter condition can be expressed as $\sigma(v, p) = a$.
- **Return Results.** The return result can be viewed as a function R that maps from the matched subgraph G' to a result set. This mapping can be a projection (returning specific vertices or edges), an aggregation (such as counting or summing), or more complex computations.

For example, suppose we have a graph G and the following GQL.

```
MATCH (n:Person)-[:KNOWS]->(m:Person)
WHERE n.age > 30
RETURN n.name, m.name
```

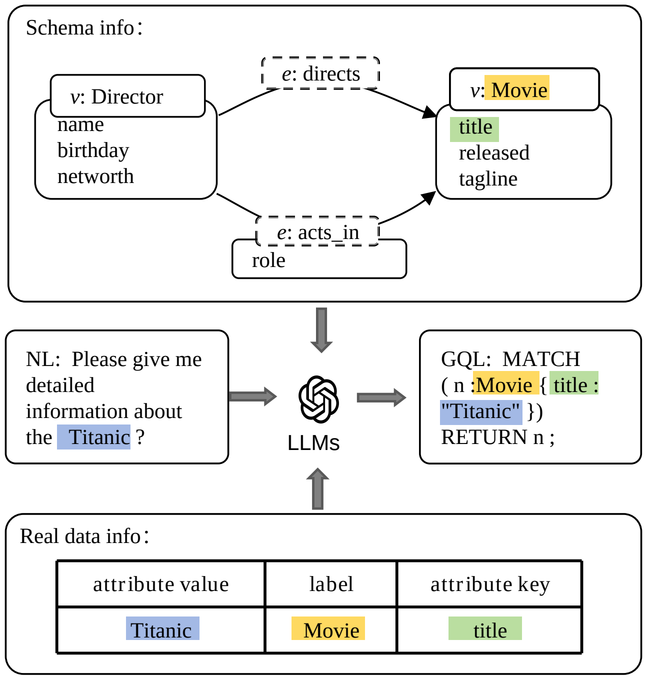
In this query, the **Pattern Matching** component Q identifies a subgraph consisting of two vertices and one edge, where both vertices are labeled *Person* and the edge is labeled *KNOWS*. The **Filter Conditions** are represented by the predicate $n.age > 30$, which ensures that only vertices with an age greater than 30 are considered. Finally, the **Return Results** specify the retrieval of the *name* attribute of the vertices n and m in the matched subgraph.

3.3 Problem Definition

Problem Statement. Converting NL to GQL via LLMs involves using schema data from GraphDBs as contextual prompts to enhance the accuracy of the generated GQL queries. Figure 3 illustrates the mapping relationship between NL and GQL.

A significant challenge in using NL queries for GraphDBs is correctly identifying whether a term corresponds to a label, a attribute key, or a attribute value. This differentiation is not always evident from the schema information alone.

If we can extract specific labels and corresponding attribute keys for the query keyword "Titanic" from real data using natural language descriptions, it can better filter out irrelevant queries, thereby improving the accuracy of query generation.



attribute value	label	attribute key
Titanic	Movie	title

Figure 3: The mapping relationship between NL and GQL

However, the majority of GraphDBs are unable to search for attribute value information using NL. To address this issue, we define attribute values that can serve as **Filter Conditions** as **FCAV**. The study proposes embedding FCAV information matching the NL query into the context for fine-tuning the LLMs. A comprehensive description of the pipeline can be found in Section 4.

4 PIPELINE

In order to enhance LLMs to generate GQL, it is essential to accurately extract query keywords from NL descriptions, particularly when these keywords correspond to attribute values in the actual data. This precision is crucial for enabling LLMs to construct more accurate pattern-matching components and filter conditions. The pipeline is illustrated in Figure 4 and comprises three main stages: (1) graph vectorization, (2) Training-Dataset generation, and (3) fine-tuning and inference processes.

4.1 Graph Vectorization

First, the vertices V and edges E of the graph G are traversed. Subsequently, the FCAV is vectorized and associated with $\lambda(v)$ (or $\lambda(e)$), along with the attribute key of the FCAV. These are stored in a vector database (V-DB), which serves as an extension to the LLM-Extended-Dataset and facilitates the generation of prompts during inference.

4.2 Training-Dataset Generation

The Training-Dataset generation process is methodically structured into four sequential steps, each designed to incrementally enhance the complexity and diversity of training datasets for LLM-based NL2GQL. The steps are as follows:

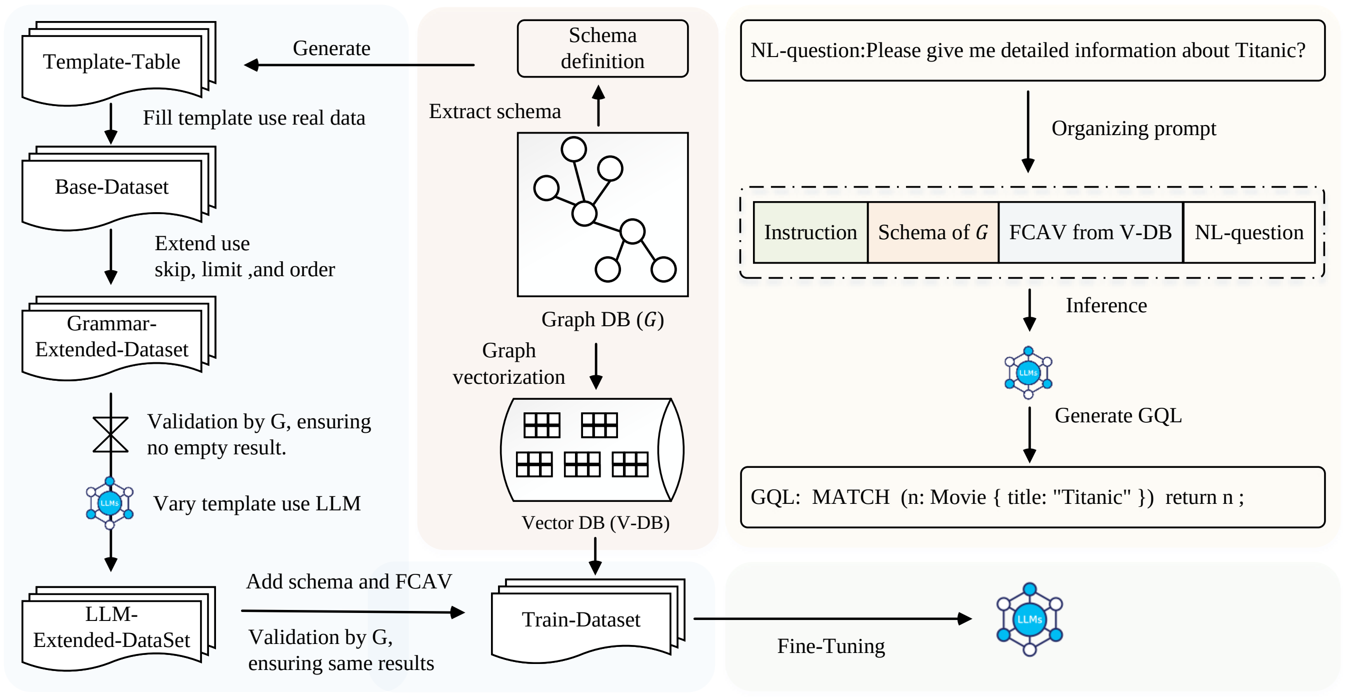


Figure 4: FCAV pipeline

Base-Dataset. The schema of G is analyzed with the objective of constructing basic NL question and GQL template pairs (NL-GQL-Pairs). Subsequently, these templates are populated with specific data by traversing G within the database, thereby generating multiple NL and GQL query pairs per template. Further details can be found in Section 5.

Grammar-Extended-Dataset. The **Base-Dataset** are extended to include pagination and sorting features, resulting in complex template structures. These templates are then populated with randomly generated numerical values. By executing GQL statements, NL-GQL-Pairs with empty results are filtered out, ultimately creating the **Grammar-Extended-Dataset**.

LLMs-Extended-Dataset. We employ LLMs to rephrase each NL-GQL-Pair in the **Grammar-Extended-Dataset** using prompt engineering techniques. This process refines the format and content of the templates by incorporating diverse expressions and syntactical variations. Subsequently, these newly generated NL-GQL-Pairs are executed in the database, and only those pairs that produce results consistent with the original GQL queries are retained. This approach enhances the robustness and diversity of the training data.

Training-Dataset. Each NL question from **LLMs-Extended-Dataset** undergoes a similarity search in the V-DB to identify relevant FCAV. By incorporating database schema information and FCAV-related details as prefixed contextual information to the NL questions. This process enables the fine-tuning of LLMs to recognize and utilize structural and real data of G, thereby facilitating the generation of accurate GQL queries.

4.3 Fine-Tuning and Inference Processes

Fine-Tuning Process. During the fine-tuning phase, the **Training-Dataset** is employed to adjust the weights of LLMs. This crucial step guarantees that the LLMs align closely with the specific knowledge in G. The fine-tuning process enables the LLMs to learn the intricate relationships among NL, the schema of G, and the actual data in G.

Inference Process. In the phase of inference, the four elements are entered into LLMs in the form of the following: ([Instruction], [Schema of G], [FCAV from V-DB], [NL question]). This prompts the LLMs to generate GQL.

5 DATASET

This section presents the methodology employed in the construction of **Training-Dataset**, with the Traditional Chinese Medicine (TCM) database serving as a case study. The data are stored using Neo4j, with the GQL being Cypher. Figure 5 illustrates the schema of the TCM database.

5.1 Database Overview

The TCM database is represented by a property graph $G = (V, E, \rho, \lambda, \sigma)$, where: V includes vertices with labels such as **Place**, **Medicine**, **Symptom**, **Function**, **Herbal**, **Categorization**, and **Excipients**. E includes edges without properties but with various relationship types. $\sigma(v, p)$ assigns properties like **name**, **type**, **price** to vertices.

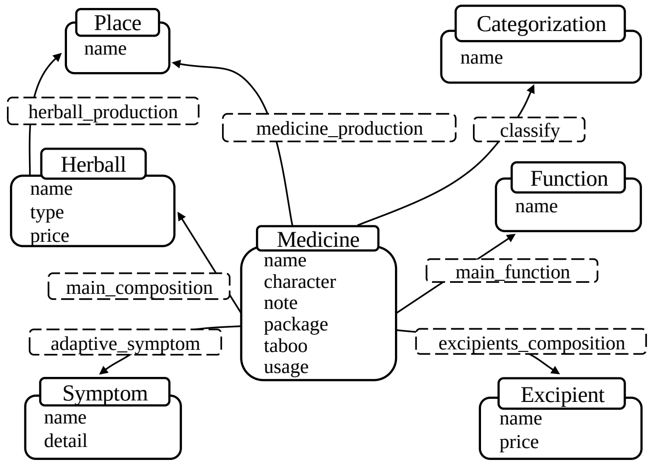


Figure 5: Schema of the TCM database

5.2 Base-Dataset Details

In order to facilitate analysis, queries are classified into two categories, as illustrated in Figure 6. The first category is **Data-Driven**, while the second is **Non-Data-Driven**. **Non-Data-Driven** queries are manually constructed (e.g., querying the total number of nodes). **Data-Driven** queries are further classified into **Schema-Filter** and **Parameter-Filter** queries, encompassing single-node and multi-hop relationships.

5.2.1 Parameter-Filter Class General Query. A **Parameter-Filter** GQL query can be described in the following form:

```
MATCH pattern_param
[
  WHERE filter_param_key_1 = filter_param_value_1
  [ AND filter_param_key_2 = filter_param_value_2 ]
  [ AND filter_param_key_3 = filter_param_value_3 ]
  ...
]
RETURN return_param
[ ORDER BY order_param ]
[ SKIP skip_param ] [ LIMIT limit_param ]
```

In the general form, ***pattern_param*** is responsible for defining the **Pattern Matching**, ***filter_param_keys*** and ***filter_param_values*** are utilized for defining the **Filter Conditions**, and ***order_param***, ***skip_param***, and ***limit_param*** are employed for controlling the optional sorting and pagination.

5.2.2 Template Filling. Algorithm 1 depicts the **Parameter-Filter** template filling algorithm, which has been designed with the specific purpose of populating templates with data extracted from graph queries. Each row in Table 1 is processed, with "DS" indicating the data source for the template. The algorithm retrieves the GQL query from the data source column and executes it on the graph G to obtain a set of results. Subsequently, a random selection mechanism, based on a predefined selection probability, is applied

Algorithm 1: Template Filling Algorithm

```
Input  : Data from template design table
Output : Filled templates
1  for each row in Table 1 do
2      gql_query ← row["DS"];
3      nl_template ← row["NT"];
4      gql_template ← row["QT"];
5      results ← executeGQL(gql_query);
6      selected_results ← empty list;
7      for each result in results do
8          if random() < selection_probability then
9              append(selected_results, result);
10     for each selected in selected_results do
11         fillTemplate(gql_template, selected) ;  /* Filling
             the pattern matching of query template */
12         fillTemplate(nl_template, selected) ;  /* Filling
             the pattern matching of natural language
             template */
13         ele_favc ← filterWithFCAV(selected);
14         combFilters ← combinations(ele_favc) ;
             /* Generate combinations of filter
             attributes */
15         fillFilterParams(combFilters);
16         all_keys ← getAllAttributeKeys(selected) ;
             /* Retrieve all attribute keys from the
             selected result */
17         key_combinations ← combinations(all_keys) ;
             /* Generate combinations of attribute
             keys */
18         fillReturnParams(key_combinations);
```

to each result, ensuring only a subset is chosen for further processing. The selected results are then used to populate both NL and GQL templates.

In the next step, attributes that can serve as FCAV are identified in each selected result. The keys and values of these attributes are extracted and combined to form filter parameters, which populate the respective templates. Additionally, all attribute keys from the selected results are retrieved and their combinations are used to fill return parameters in the templates.

This multi-step process ensures that the templates are comprehensively populated with relevant data. Once the templates are populated, the dataset construction is completed in accordance with the procedures outlined in Section 4.2. The **Schema-Filter** process, like the **Parameter-Filter**, is conducted with the objective of ensuring the quality and integrity of the data.

6 EXPERIMENTS

6.1 Experimental Setup

The LLaMA-Factory tool [30] was employed in conjunction with LoRA to conduct a fine-tuning process on the LLMs. The training procedure utilized a linear scheduler with an initial learning rate of

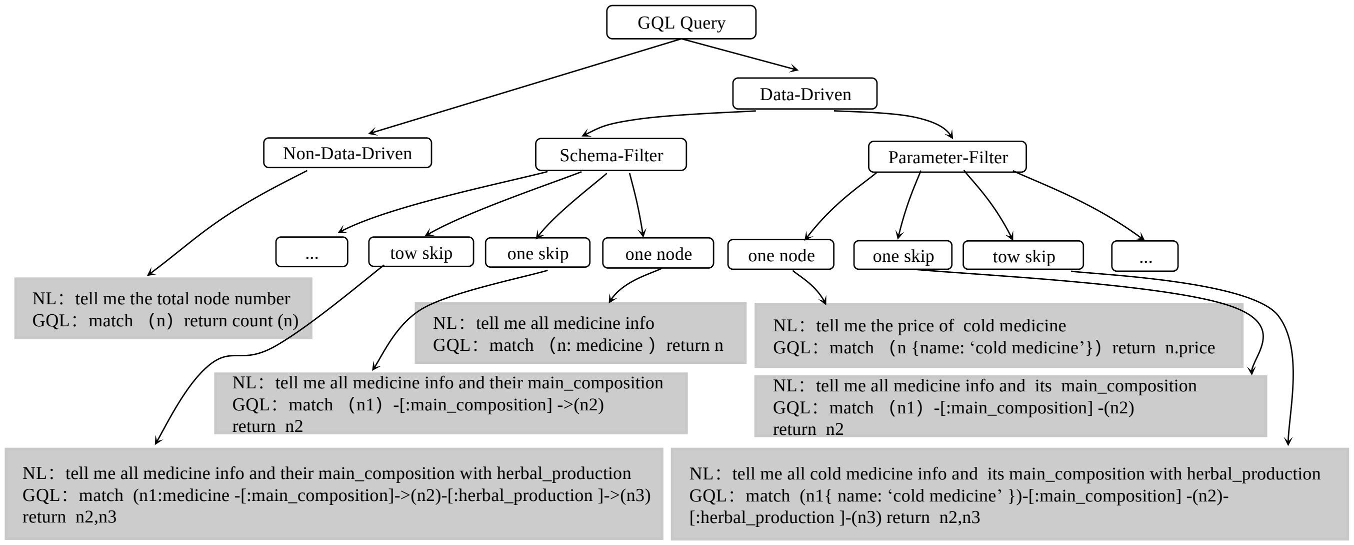

GQL Query
Data-Driven
Non-Data-Driven
Schema-Filter
Parameter-Filter
...
tow skip
one skip
one node
one node
one skip
tow skip
...
NL： tell me the total node number
GQL： match （n） return count (n)
NL： tell me all medicine info
GQL： match （n: medicine ） return n
NL： tell me the price of cold medicine
GQL： match （n {name: 'cold medicine'}） return n.price
NL： tell me all medicine info and their main_composition
GQL： match （n1） -[:main_composition] ->(n2) return n2
NL： tell me all medicine info and its main_composition
GQL： match （n1） -[:main_composition] -(n2) return n2
NL： tell me all medicine info and their main_composition with herbal_production
GQL： match (n1:medicine -[:main_composition]->(n2)-[:herbal_production]->(n3) return n2,n3
NL： tell me all cold medicine info and its main_composition with herbal_production
GQL： match (n1{ name: 'cold medicine' })-[:main_composition] -(n2)-[:herbal_production]-(n3) return n2,n3

Figure 6: Classification of query templates

Table 1: Template Design And Data Source Samples

Types	Data Source of Template (DS)	NL Template (NT)	GQL Template (QT)
`One node`	`MATCH (n1) RETURN n1;`	`query filter_param_value then return return_param`	`MATCH (n1) where n1.filter_param_key = filter_param_value RETURN return_param`
`One skip`	`MATCH (n1)-[r1]->(n2) RETURN *;`	`query` λ`(n2) of` λ`(r1) filter_param_value then return return_param`	`MATCH (n1:`λ`(n1))-[r1:`λ`(r2)] ->(n2:`λ`(n2)) where n1.filter_param_key = filter_param_value RETURN return_param`
`Two skip`	`MATCH (n1)-[r1]->(n2) -[r2]->(n3) RETURN *;`	`query` λ`(n3) of` λ`(r2) of` λ`(n2) of` λ`(r1) of filter_param_value then return return_param`	`MATCH (n1:`λ`(n1))-[r1:`λ`(r1)] ->(n2:`λ`(n2))-[r2:`λ`(r2)] ->(n3:`λ`(n3)) where n1.filter_param_key = filter_param_value RETURN return_param`

$1e-4$. The training parameter for epochs was set to 3. The AdamW optimizer was applied, and the batch size for training was set to 1. All experiments were conducted using an Nvidia GeForce 4090 GPU. During the experiments, we used BCEmbedding[22] as the FCAV vectorization model and pgvector[26] as the V-DB.

Three models were selected for these experiments, each distinguished by its proficiency in specific areas of NLP. The first model, ChatGLM3-6B [27], is optimized for Chinese conversational generation and comprises 6 billion parameters. The second model, Llama3-8B [28], is a versatile model with 8 billion parameters, suitable for a variety of NLP tasks. The third model, Qwen-7B [1], has 7 billion parameters and is recognized for its strong performance across multiple language modeling tasks.

In our experiments, four distinct prompts were devised and evaluated on both the non-fine-tuned and fine-tuned models using these specific prompts. This comprehensive approach aims to encompass all current baseline methodologies for generating GQL using LLMs. The experimental prompt settings are detailed in Table 2.

Table 2: Prompt Types and Descriptions

Prompt	Form of Input
NP	([Instruction] [NL question])
SP	([Instruction] [Schema of G] [NL question])
SGP (Ours)	([Instruction] [Schema of G] [FCAV from V-DB] [NL question])
GP	([Instruction] [FCAV from V-DB] [NL question])

6.2 Evaluation Metric

To assess the performance of GQL queries generated by various LLMs-based methods, we used three evaluation metrics: syntactic accuracy (SyA), semantic accuracy (SeA), and exact match rate (EMR).

(1) Syntactic Accuracy (SyA)

SyA is defined as the ratio of the number of syntactically correct GQL queries to the total number of generated queries:

$$\text{SyA} = \frac{N_{\text{correct syntax}}}{N_{\text{total queries}}} \quad (1)$$

Where $N_{\text{correct syntax}}$ denotes the number of GQL queries that are syntactically correct (i.e., queries that do not produce syntax errors when executed in the database), and $N_{\text{total queries}}$ represents the total number of generated GQL queries.

(2) Semantic Accuracy (SeA)

SeA measures the proportion of GQL queries that yield results consistent with the expected outcomes to the total number of generated queries:

$$\text{SeA} = \frac{N_{\text{correct results}}}{N_{\text{total queries}}} \quad (2)$$

Where $N_{\text{correct results}}$ denotes the number of GQL queries that produce the expected results. The SeA is determined by manually evaluating the generated results against the expected outputs from the test set. A query is considered semantically correct if its output contains the results desired by the test set.

(3) Exact Match Rate (EMR)

EMR is defined as the ratio of the number of generated GQL queries that exactly match the reference queries in the dataset to the total number of generated queries:

$$\text{EMR} = \frac{N_{\text{exact match}}}{N_{\text{total queries}}} \quad (3)$$

Where $N_{\text{exact match}}$ denotes the number of generated GQL queries that exactly match the standard reference queries. To calculate the EMR, all spaces are removed, and all characters are converted to lowercase before matching.

6.3 Performance of Base Models

Initial tests were conducted without fine-tuning to establish a performance baseline. The results, summarized in Table 3, highlight the NL understanding capabilities of each model under different prompting strategies.

Table 3: Base Model Performance

Model	Method	SyA (%)	SeA (%)	EMR (%)
ChatGLM3-6B	NP	17.65	0.65	0.00
	SP	25.82	5.88	0.00
	SGP (Ours)	**32.03**	**2.61**	0.00
Llama3-8B	NP	15.69	0.98	0.00
	SP	23.86	2.94	0.65
	SGP (Ours)	**28.43**	**5.88**	**0.98**
Qwen-7B	NP	**58.50**	1.63	0.00
	SP	7.19	0.98	0.00
	SGP (Ours)	9.80	**2.94**	0.00

In comparison to SP, our approach on ChatGLM3 and Llama3 generated a greater number of usable GQL queries due to the incorporation of FCAV context. In the case of the Qwen model, the incorporation of contextual data had the effect of reducing the probability of generating a usable GQL. This discrepancy may be attributed to the fact that the Qwen-7B model had been trained with GQL capabilities, whereas our prompt format differs from the training data. Nevertheless, the SGP still demonstrates superior performance when compared to SP contexts.

6.4 Fine-tuned Model Performance

Overall, while LLM models without fine-tuning exhibit some capacity for GQL generation, they remain constrained in their practical applicability.

Table 4: Fine-tuned Model Benchmark Result

Model	Method	SyA (%)	SeA (%)	EMR (%)
ChatGLM3-6B	NP	34.92	7.93	1.90
	SP	93.65	49.90	40.32
	SGP (Ours)	**94.60**	**53.34**	**42.54**
Llama3-8B	NP	99.05	77.46	56.83
	SP	99.37	88.57	59.27
	SGP (Ours)	**99.47**	**89.52**	**60.32**
Qwen-7B	NP	96.51	68.25	44.76
	SP	97.78	78.09	54.29
	SGP (Ours)	**98.73**	**78.24**	**55.87**

Following the application of fine-tuning techniques, each model exhibited notable enhancements across all performance metrics. These improvements are detailed in Table 4, which presents the benchmark results for the fine-tuned models.

Our method demonstrated superior performance compared to the baseline SP in both SyA and SeA metrics across all models, substantiating its effectiveness. The Llama3 model shows superior performance across all metrics, indicating its suitability for our dataset with LoRA fine-tuning. The LlAMA3 and Qwen models achieved 78% and 89% semantic correctness, respectively, using our method, indicating its practical applicability.

Figure 7 synthesizes the results of the first two experiments. In the figure, models without fine-tuning are denoted by the suffix 'U', while fine-tuned models are denoted by 'F'. The performance of fine-tuned models consistently surpasses that of non-fine-tuned models across all inference tests. Furthermore, among the fine-tuning methods, our SGP approach consistently achieves optimal performance across various models.

6.5 Ablation Study

An ablation study was conducted to evaluate the contribution of each component in our method. The results are presented in Table 5.

The experimental results demonstrate that LLMs produce the most accurate grammars when using GP, while the highest rate of correct semantics is achieved through SGP. These findings indicate that incorporating contextual information of FCAV from real data facilitates the generation of accurate grammar. However, accurate semantic understanding still requires the incorporation of database schema information.

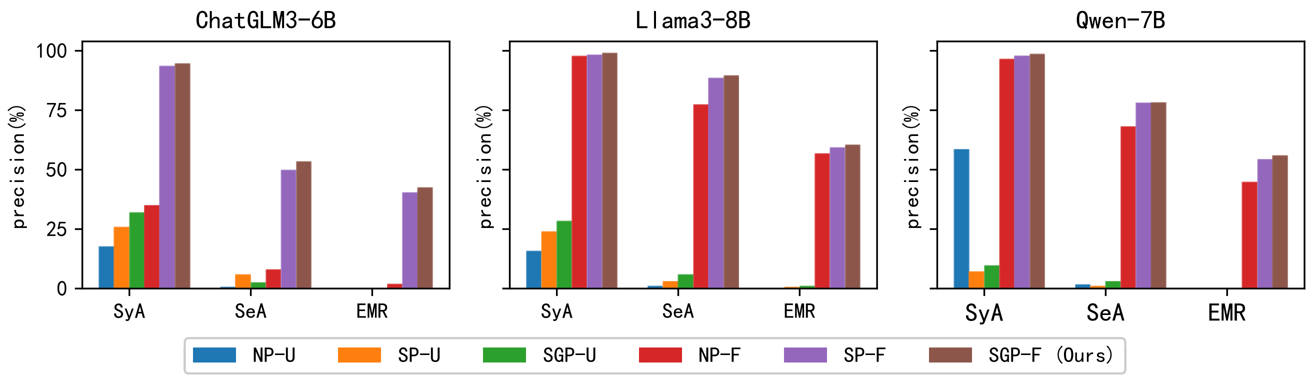


Figure 7: Synthesizing the results

Table 5: Ablation Study Result

Model	Method	SyA (%)	SeA (%)	EMR (%)
ChatGLM3-6B	SP	93.65	49.90	40.32
	GP	**95.24**	40.89	32.38
	SGP (Ours)	94.60	**53.34**	**42.54**
Llama3-8B	SP	99.37	88.57	59.27
	GP	98.41	84.12	56.51
	SGP (Ours)	**99.47**	**89.52**	**60.32**
Qwen-7B	SP	97.78	78.09	54.29
	GP	**99.37**	72.06	53.33
	SGP (Ours)	98.73	**78.24**	**55.87**

6.6 Domain Migration Experiment

The objective of this experiment is to assess the robustness and adaptability of our fine-tuning methodology across different domains using the EduGQL dataset.

Table 6: Domain Migration Experiment Results

Model	Method	SyA (%)	SeA (%)	EMR (%)
ChatGLM3-6B	SP	84.99	7.60	0
	SGP (Ours)	**86.77**	**10.27**	0
Llama3-8B	SP	**98.47**	41.50	0
	SGP (Ours)	**98.47**	**43.47**	0
Qwen-7B	SP	**97.46**	29.47	0
	SGP (Ours)	91.60	**32.44**	0

The results show that each model exhibits a decline in grammatical accuracy (between 2% and 10%) and semantic understanding (approximately 46%) compared to their performance in the TCM domain. Notably, the SeA achieved through the SGP method still surpasses that of the SP method by 2 ∼ 3%.

7 LIMITATIONS

Our work is applicable to property graphs and object-oriented modeling methods. While property graph models are widely used, there are various graph database modeling approaches in practice. In some scenarios, each entity is represented as a distinct label, which our methodology does not fully accommodate. This limitation arises because our approach focuses on constructing query entities based on attribute values during dataset creation. Consequently, our methodology may be less effective in environments where entities are identified by labels rather than attributes.

8 CONCLUSION

This paper presents a robust framework for transforming NL queries into GQL queries for GraphDBs using LLMs. Experimental results demonstrate that our approach outperforms existing methods in SyA, SeA, and EMR metrics and shows superior performance in domain migration scenarios. These findings highlight the potential of our framework to improve GQL generation across diverse domains, establishing a solid foundation for broader applications.

ACKNOWLEDGMENTS

This work is supported by HUAWEI.

REFERENCES

[1] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[2] Bradley R Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Mike Personick, Karthik Rajan, et al. 2018. Amazon Neptune: Graph Data Management in the Cloud.. In *ISWC (P&D/Industry/BlueSky)*.

[3] Shambhavi Chauhan and Deepak Arora. 2023. Development of Cloud based Smart Voice Assistant using Amazon Web Services. In *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*. 1217–1221. https://doi.org/10.1109/ICAAIC56838.2023.10140284

[4] Hafsa Shareef Dar, M Ikramullah Lali, Moin Ul Din, Khalid Mahmood Malik, and Syed Ahmad Chan Bukhari. 2019. Frameworks for querying databases using natural language: a literature review. *arXiv preprint arXiv:1909.01822* (2019).

[5] Mauro Dragoni, Monika Solanki, and Eva Blomqvist. 2017. *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017, Portoroz, Slovenia, May 28-June 1, 2017, Revised Selected Papers*. Vol. 769. Springer.

[6] Guandong Feng, Guoliang Zhu, Shengze Shi, Yue Sun, Zhongyi Fan, Sulin Gao, and Jun Hu. 2023. Robust NL-to-Cypher Translation for KBQA: Harnessing Large Language Model with Chain of Prompts. In *China Conference on Knowledge Graph and Semantic Computing*. Springer, 317–326.

[7] Emilio Ferrara. 2023. Should chatgpt be biased? challenges and risks of bias in large language models. *arXiv preprint arXiv:2304.03738* (2023).

[8] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and

Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 international conference on management of data*. 1433–1445.

[9] Anand Gokul. 2023. Llms and ai: Understanding its reach and impact. (2023).

[10] Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. Spcql: A semantic parsing dataset for converting natural language into cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3973–3977.

[11] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[12] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169* (2023).

[13] Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. In *International Conference on Machine Learning*. PMLR, 15696–15707.

[14] Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. 2023. Knowledge graph-augmented language models for knowledge-grounded dialogue generation. *arXiv preprint arXiv:2305.18846* (2023).

[15] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Complex knowledge base question answering: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2022), 11196–11215.

[16] Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2024. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 18608–18616.

[17] Yuanyuan Liang, Keren Tan, Tingyu Xie, Wenbiao Tao, Siyuan Wang, Yunshi Lan, and Weining Qian. 2024. Aligning Large Language Models to a Domain-specific Graph Database. *arXiv preprint arXiv:2402.16567* (2024).

[18] Haoran Luo, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, et al. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975* (2023).

[19] Lars-Peter Meyer, Claus Stadler, Johannes Frey, Norman Radtke, Kurt Junghanns, Roy Meissner, Gordian Dziwis, Kirill Bulert, and Michael Martin. 2023. Llm-assisted knowledge graph engineering: Experiments with chatgpt. In *Working conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow*. Springer Fachmedien Wiesbaden Wiesbaden, 103–115.

[20] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, Vol. 2324. 141–147.

[21] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[22] Inc. NetEase Youdao. 2023. BCEmbedding: Bilingual and Crosslingual Embedding for RAG. https://github.com/netease-youdao/BCEmbedding.

[23] Ermelinda Oro and Massimo Ruffolo. 2015. A natural language interface for querying RDF and graph databases. *Consiglio Nazionale delle Ricerche Istituto di Calcoloe Reti and Alte Prestazioni* (2015).

[24] Fatma Õzcan, Abdul Quamar, Jaydeep Sen, Chuan Lei, and Vasilis Efthymiou. 2020. State of the art and open challenges in natural language interfaces to data. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2629–2636.

[25] Anand Panchbhai, Tommaso Soru, and Edgard Marx. 2020. Exploring sequence-to-sequence models for SPARQL pattern composition. In *Knowledge Graphs and Semantic Web: Second Iberoamerican Conference and First Indo-American Conference, KGSWC 2020, Mérida, Mexico, November 26–27, 2020, Proceedings 2*. Springer, 158–165.

[26] pgvector. [n.d.]. pgvector: Open-source extension for vector similarity search in PostgreSQL. https://github.com/pgvector/pgvector.

[27] GLM Team, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *arXiv e-prints* (2024), arXiv–2406.

[28] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[29] Min Wu, Xinglu Yi, Hui Yu, Yu Liu, and Yujue Wang. 2022. Nebula Graph: An open source distributed graph database. *arXiv preprint arXiv:2206.07278* (2022).

[30] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, Bangkok, Thailand. http://arxiv.org/abs/2403.13372

[31] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

[32] Ziije Zhong, Linqing Zhong, Zhaoze Sun, Qingyun Jin, Zengchang Qin, and Xiaofan Zhang. 2024. SyntheT2C: Generating Synthetic Data for Fine-Tuning Large Language Models on the Text2Cypher Task. *arXiv preprint arXiv:2406.10710* (2024).

[33] Yuhang Zhou, He Yu, Siyu Tian, Dan Chen, Liuzhi Zhou, Xinlin Yu, Chuanjun Ji, Sen Liu, Guangnan Ye, and Hongfeng Chai. 2023. R^3-NL2GQL: A Hybrid Models Approach for for Accuracy Enhancing and Hallucinations Mitigation. *arXiv preprint arXiv:2311.01862* (2023).