Multi-turn Natural Language to Graph Query Language Translation

Anonymous ACL submission

Abstract

In recent years, research on transforming natural language into graph query language (NL2GQL) has been increasing. Most existing methods focus on single-turn transformation from NL to GQL. In practical applications, user interactions with graph databases are typically multi-turn, dynamic, and context-dependent. While single-turn methods can handle straightforward queries, more complex scenarios often require users to iteratively adjust their queries, investigate the connections between entities, or request additional details across multiple dialogue turns. Research focused on single-turn conversion fails to effectively address multi-turn dialogues and complex context dependencies. Additionally, the scarcity of high-quality multi-turn NL2GQL datasets further hinders the progress of this field. To address this challenge, we propose an automated method for constructing multi-turn NL2GQL datasets based on Large Language Models (LLMs) , and apply this method to develop the **MTGQL** dataset, which is constructed from a financial market graph database and will be publicly released for future research. Moreover, we propose three types of baseline methods to assess the effectiveness of multi-turn NL2GQL translation, thereby laying a solid foundation for future research.

1 Introduction

As data complexity and interconnectedness grow across various domains, graph data structures have become essential for effectively representing and analyzing relationships (Zhao et al., 2022a; Sui et al., 2024). This increasing demand for efficient data representation has driven the widespread adoption of graph databases. Consequently, graph query language (GQL) has emerged as a crucial tool for interacting with these systems, playing a pivotal role in tasks such as database management, information retrieval, and data analysis (Lopes et al., 2023; Wang et al., 2020; Pavliš, 2024), as shown in Figure 1. However, translating natural language (NL) queries into GQL presents a significant challenge, as it requires users to possess technical expertise in database operations and a deep understanding of specific query syntax and patterns. This complexity creates a substantial barrier for individuals without a technical background (Zhao et al., 2022b, 2023). To address this challenge, numerous automatic NL2GQL methods have been proposed (Guo et al., 2022; Zhou et al., 2024b; Liang et al., 2024a; Tao et al., 2024; Tran et al., 2024), making graph databases accessible to more audiences.

Recent advances in NL2GQL are primarily derived from the Seq2Seq framework, such as those demonstrated in (Guo et al., 2022) and CoBGT (Tran et al., 2024). With the rise of LLMs, performance has been further enhanced, leading to the development of numerous LLM-based methods (Zhou et al., 2024b; Liang et al., 2024a; Tao et al., 2024; Liang et al., 2024b; Liu et al., 2024). Alongside these methods, several NL2GQL datasets have been developed, including SpCQL (Guo et al., 2022), CySpider (Zhao et al., 2023), Text2Cypher (Ozsoy et al., 2024), R^3-NL2GQL(Zhou et al., 2024b), TCMGQL, EduGQL(Liu et al., 2024), and StockGQL (Liang et al., 2024b). The proposed methods and datasets mainly focus on single-turn queries.

While single-turn NL2GQL translation can handle relatively simple queries, multi-turn interactions introduce several complexities that require advanced handling. First, the system must maintain context across multiple historical queries, as each new query builds upon the information provided in previous ones. This necessitates robust context management to accurately capture the user's evolving intent and ensure the generation of consistent, relevant queries. Second, as users refine or expand their queries during the interaction, the system must dynamically adjust the context to ac-

User: Which securities stock **opened at the highest price** today?
System: **CITIC Securities.**
(GQL: match (s:stock)-[:belong_to]->(i:industry) WHERE i.name = 'securities' return s.name order by s.**opening_price** desc limit 1)
User: What **price**?
System: ¥30.26
(GQL: match (s:stock {name: '**CITIC Securities**'})-[:has_data]->(d:stock_data {date: '2025-01-08'}) return d.**opening_price**)
User: And yesterday?
System: ¥36.25
(GQL: match (s:stock {name: '**CITIC Securities**'})-[:has_data]->(d:stock_data {date: '2025-01-07'}) return d.**opening_price**)
User: How about **Guotai Junan**?
System: ¥20.00
(GQL: match (s:stock {name: ' **Guotai Junan Securities**'})-[:has_data]->(d:stock_data {date: '2025-01-08'}) return d.**opening_price**)

Figure 1: An example of a multi-turn interaction between a **User** and a **System**, with the orange sections representing the corresponding Cypher-based GQL for each question. The color coding highlights the contextual dependencies, such as **opening price** , **CITIC Securities** and **Guotai Junan Securities**.

commodate these changes. Last but not least, current datasets are primarily designed for single-turn queries, resulting in limited data available for training and evaluating multi-turn systems. This constraint hampers the development of more sophisticated, context-aware solutions.

To tackle the challenge posed by the scarcity of multi-turn NL2GQL datasets, we propose a **dependency-aware multi-turn dataset construction framework**, which performs collaborative optimization between LLMs, graph data, and dialogue dependency in an iterative way. Our framework is composed of four essential components: a Context Manager, Question Generator, GQL Generator, and GQL Optimizer. Here, context manager plays as a central unit to integrate the information of dialogue history and graph data and send to other constituents. Question generator, GQL generator, and GQL optimizer are LLM-based constituents to analysis the information from the context manager and output the generated questions, GQLs, and answers. They also interact with each other for mutual checking and correction. Using this framework, we have created the MTGQL dataset, a Chinese multi-turn NL2GQL dataset based on a financial market NebulaGraph database.

Our main contributions are as follows:

- **A Standard Framework:** We propose a novel framework for constructing multi-turn NL2GQL datasets. To the best of our knowledge, this is the first method specifically designed for building such datasets.
- **MTGQL Dataset:** By applying our approach to a Chinese financial NebulaGraph database, we built MTGQL, the first Chinese multi-turn NL2GQL dataset.
- **Benchmark Methods:** We introduce the Backmarch methods for the MTGQL dataset, establishing a strong foundation for future research.

2 Related Work

2.1 NL2GQL

Early work in NL2GQL focused on template generation and heuristic rule-based systems. Recent advancements in NL2GQL tasks have seen a shift to deep learning-based approaches. Among the pioneering studies, the work (Guo et al., 2022) was the first to apply a Seq2Seq framework to NL2GQL, introducing a copying mechanism alongside the Seq2Seq model to enhance GQL generation. This approach paved the way for subsequent deep learning-based models in this space. The CoBGT model (Tran et al., 2024) further advanced this field by integrating key-value extraction, relation-property prediction, and Cypher query generation. This model combines BERT, GraphSAGE, and Transformer architectures to address the NL2GQL task.

The emergence of LLMs has further advanced the research in NL2GQL. The paper (Tao et al.,

2024) presented a revision-based method for NL2GQL, leveraging LLMs without fine-tuning, further simplifying the process of adapting LLMs for NL2GQL tasks. R^3-NL2GQL (Zhou et al., 2024b) integrates small and large foundation models for ranking, rewriting, and refining tasks, enhancing query quality by better understanding context and relationships. The work in (Liang et al., 2024a) proposed aligning LLMs with domain-specific graph databases to enhance query accuracy and domain relevance. It emphasizes the adaptability of LLMs when tailored to specific graph schemas, ensuring that generated queries are contextually appropriate. In another study, (Liang et al., 2024b) proposed a three-agent system for NL2GQL, comprising a Preprocessor for data handling, a Generator for GQL creation, and a Refiner that refines queries based on execution results. This multi-agent approach provides a more structured and efficient translation process, addressing both query generation and validation. The method (Liu et al., 2024) proposed using template-filling and problem rewriting techniques with LLMs to provide contextual information, improving the model's comprehension of the complex relationships between NL, graph schemas, and database data. These methods are all based on the single-turn NL2GQL task[1].

2.2 NL2GQL Dataset

The development of NL2GQL datasets has also evolved alongside advances in model architectures. Several datasets have been proposed in recent years, each addressing different aspects of the NL2GQL task. The SpCQL (Guo et al., 2022) dataset is constructed by manually annotating 10,000 NL queries with corresponding Cypher queries based on a single Neo4j graph database. CySpider (Zhao et al., 2023) dataset is constructed by developing a SQL2Cypher algorithm that maps SQL queries to Cypher clauses, which are then paired with the original natural language queries to create a parallel corpus. Text2Cypher (Ozsoy et al., 2024) combined, cleaned, and organized several publicly available datasets into a total of 44,387 instances to enable effective fine-tuning and evaluation. R^3-NL2GQL (Zhou et al., 2024b) constructed the dataset by manually creating NL-GQL pairs, using foundation models to generate diverse interpretations, and refining them manually.

[1]A more detailed comparison with similar tasks is provided in the Appendix 9.1.

Recently, using LLMs to construct data has become an effective solution to the problem of data scarcity, especially for tasks in specific domains (Ding et al., 2024; Long et al., 2024; Zhou et al., 2024a). The TCMGQL and EduGQL (Liu et al., 2024) datasets were constructed from real-world databases, ensuring standardized types and diversity. Over ten NL and GQL templates were developed based on database schema information, further enhanced by LLMs. The work (Liang et al., 2024a) constructs datasets by first generating NL-GQL pairs from a graph database, followed by a two-step data augmentation process using ChatGPT to ensure diverse and comprehensive query coverage. The generated pairs are then grounded and verified. Building upon the work in (Liang et al., 2024a), the work (Liang et al., 2024b) introduced improvements by incorporating subgraph extraction related to GQL and the colloquialization of named entities, while also constructing the StockGQL dataset. Unlike these methods, we focus on developing a multi-turn NL2GQL dataset.

3 Multi-turn NL2GQL Task Formulation

A graph database G consists of a large number of connected data (nodes and edges).

We first define single-turn NL2GQL as follows. Given a graph database G and a question Q, the NL2GQL system is supposed to return an executable GQL command that can be executed against G and produce an answer A:

$$GQL_t = \mathbb{F}(Q, G).$$

Here, $\mathbb{F}$ is a function that generates the graph query language GQL based on Q, and G. In single-turn NL2GQL, different question-answer pairs in the dataset $\mathcal{D} = \{(Q_1, A_1), (Q_2, A_2), ...\}$ are independent.

In comparison, the interdependent question-answer pairs in multi-turn NL2GQL problem form a complete dialogue, denoted as $C = ((Q_1, A_1), (Q_2, A_2), ..., (Q_m, A_m))$ and a set of dialogues forms a dataset $\mathcal{D} = \{C_1, C_2, ...\}$. We refer to each question-answer pair as *one round of the dialogue*. In the multi-turn NL2GQL, at the t-th round, given multiple rounds of historical interaction between the user C_t, the objective is to generate the GQL, denoted as GQL_t, corresponding to the question Q_t:

$$GQL_t = \mathbb{F}(Q_t, C_t, G),$$

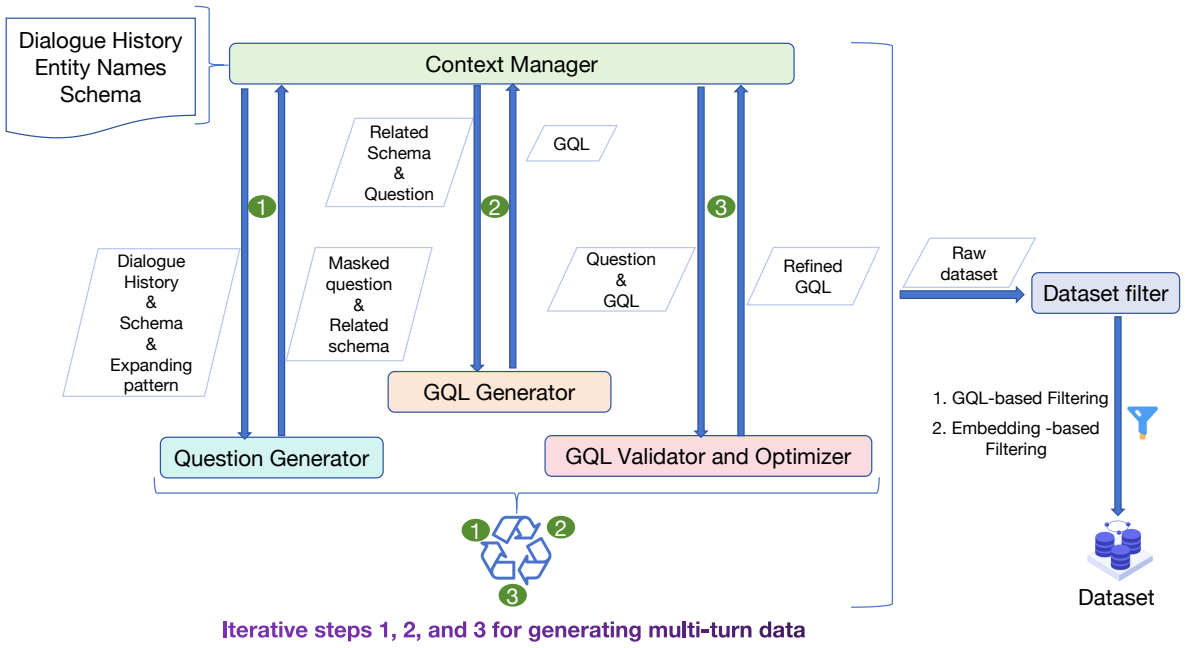

Figure 2: Our framework consists of five synergistic components: the Context Manager, Question Generator, GQL Generator, GQL Validator and Optimizer, and Dataset Filter. These components work collaboratively to handle question generation, GQL generation, GQL validation and refinement, and dataset filtering. Steps 1, 2, and 3 are iteratively executed for each data point to generate multi-turn data.

where $C_t = \{Q_1, A_1, ..., Q_{(t-1)}, A_{(t-1)}\}$ includes all relevant user inputs and system responses executed against G via the GQLs.

4 A Dependency-aware Multi-turn Dataset Construction Framework

4.1 Overview

To construct a multi-turn NL2GQL dataset, we follow three key criteria that distinguish it from single-turn NL2GQL: (1) **Graph Grounding:** Each question should be factually grounded via G to ensure its corresponding answers can be successfully retrieved from the graph data with a GQL. (2) **Interdependent Turns:** The question-answer pairs in a dialogue should be interdependent. Specifically, the question in the current round could be linked to the dialogue history via either questions or answers in the previous rounds. (3) **Diverse Dependency Types:** The types of the questions and dialogue dependencies should present diversity to cover the application of practical scenario.

As showed in Figure 2, the framework comprises five interconnected components: **Context Manager**, **Question Generator**, **GQL Generator**, **GQL Validator and Optimizer**, and **Dataset Filter**. Next, we will detail the implementation and role of each core component.

4.2 Context Manager

The Context Manager is the control components of the system, Its functions include the following aspects:

Updating the Dialogue History: The Context Manager is responsible for maintaining the dialogue history, which includes C_t, the set of entities and relations, and the expansion pattern history. It continuously updates the dialogue history to ensure that all interactions are accurately tracked.

Fulfilling Masked Questions: Since the Question Generator generates specific entity names for certain questions but may not have access to the available entities in the database, placeholders are used. Therefore, another responsibility of the Context Manager is to replace the placeholders with actual entity names from the graph database.

Controlling the Generation Process: The Context Manager oversees the entire data generation process, controlling both the start and end. It is also responsible for selecting question expansion patterns based on the set of entities and relations in the history. To ensure the generation of high-quality questions, we have designed six fundamental expansion patterns, as shown in Table 1, and the expansion pattern selection algorithm is detailed in Appendix 9.2. We adjust the number of conversation rounds iteratively, keeping the total rounds per

Pattern	Description	Example
P1: Attribute Follow-up	Generates follow-up questions about an entity's attributes based on the previous query.	Q1: What is the largest stock in the liquor industry? A1: Moutai. Q2: What is the registered capital?
P2: Temporal Shift	Introduces the time dimension to generate queries related to historical data.	Q1: What is the highest price of Moutai today? A1: 20.5 Q2: What was the closing price yesterday?
P3: Relation Extension	Expands the dialogue by querying related relationships.	Q1: What is the stock code for Tencent? A1: HK0700 Q2: What is the industry data?
P4: Same-Type Entity	Used for comparative reasoning between multiple entities.	Q1: What is the opening price of Baidu today? A1: 150 Q2: What about Alibaba?
P5: Aggregation Calculation	Involves queries requiring aggregation calculations such as averages or sums.	Q1: What is the opening price of Tengfei today? A1: 417 Q3: What is the day-on-day growth?
P6: Conditional Filtering	Filters data based on specific conditions.	Q1: Which funds have a management fee below 1%? A1: Fund A, Fund B Q2: Which ones have a size greater than 5 billion?

Table 1: Patterns for expanding subsequent questions.

data point between 5 and 8 to maintain appropriate depth and complexity.

4.3 Question Generator

We use an LLM as the Question Generator, categorizing questions into initial and follow-up types. The initial question is randomly generated based on the schema of G, while subsequent questions follow the expansion patterns from the Context Manager. These questions must inherit context, promoting diversity, complexity, and a colloquial tone.

To better guide the LLM in generating high-quality questions, **we instruct it to produce more colloquial, informal, and ambiguous expressions that more accurately simulate real user queries.** The prompt format is shown in Appendix 9.3. It is important to note that since the Question Generator is only aware of the schema of G and does not have access to the specific entities stored within the database, questions involving entities are generated as placeholder templates. For example, `What is the opening price of [s] stock today?` where `[s]` represents a placeholder for the stock entity name.

4.4 GQL Generator

The GQL Generator is responsible for generating the corresponding GQL based on the schema of G and the complete question provided by the Context Manager. To enhance generation efficiency, we use the full schema to construct the prompt for fine-tuning the LLM, as outlined in Paper (Liang et al., 2024a). With the fine-tuned LLM, the GQL Generator ensures accurate understanding and handling of the graph database's schema when generating GQL.

4.5 GQL Validator and Optimizer

The GQL Validator and Optimizer play a crucial role in ensuring that the GQL are both syntactically and semantically correct. The workflow of the GQL Validator and Optimizer proceeds as follows: first, Syntax Validation, followed by Semantic Validation. Only GQLs containing syntax or semantic errors will undergo optimization for improvement.

Syntax Validation: This ensures that the generated GQL statements are syntactically correct and executable in the graph database. The GQL is executed on the database, and if it runs successfully with expected results, it is syntactically correct; otherwise, it is flagged for optimization.

Semantic Validation: This ensures that the GQL accurately reflects the original question's intent. We utilize the reverse generation validation method introduced in paper (Liang et al., 2024a) to infer the original question from the generated GQL. If the vector embedding similarity between the inferred and original question is low, it indicates that the generated GQL requires further optimization.

GQL Optimization: When syntax errors are detected, the system combines the original ques-

tion, generated GQL, and error information into a prompt for the LLM to correct. The modified GQL is then re-validated for syntax. For semantic optimization, if the GQL doesn't align with the original question's intent, both the question and GQL are input into the LLM for correction. The corrected GQL undergoes semantic validation, and this process repeats up to three times. If all attempts fail, the system instructs the Context Manager to regenerate the question.

4.6 Dataset Filter

After dataset generation, while the methods outlined above ensure the quality of each data point, they cannot guarantee the absence of similarity and redundancy. To address this, we apply two filtering methods.

GQL-based Filtering: We replace entity names in the GQL with placeholders and collect the masked GQL into a set. By comparing sets across data points, we calculate their similarity. If more than three identical masked GQL are found, one is discarded as redundant, effectively reducing duplicates in the dataset.

Embedding-based Filtering: To prevent high similarity between questions across data points, we concatenate all questions within each data entry and encode them using the `all-MiniLM-L6-v2` model from Sentence-BERT to obtain high-dimensional semantic representations. We then compute the cosine similarity between these vector embeddings across all data points. Any data point pair with cosine similarity exceeding a threshold of 0.6 is considered semantically redundant and discarded.

Finally, we applied our approach to a Chinese financial market NebulaGraph database to develop the MTGQL dataset based on nGQL syntax.

5 Data Analysis

5.1 Dataset Statistics

As shown in Table 3, the dataset contains 4,500 multi-turn dialogues, split into 3,000 for training, 500 for development, and 1,000 for testing. Each dialogue has an average of 6.49 turns, reflecting balanced dialogue depth. In total, there are 29,196 GQL statements, with multiple queries per dialogue, indicating the dataset's complexity. On average, each dialogue involves 4.79 entities and 5.59 relations, requiring models to handle rich and diverse graph structures. The slightly higher averages in the test set suggest a more challenging evaluation setting. Overall, the dataset is well-structured and suitable for training and evaluating models on dialogue-based graph query tasks.

5.2 Human Evaluation

We evaluated the quality of the dataset by asking three domain experts to rate 200 randomly selected dialogues from each of the training, validation, and test sets. The evaluation focused on four dimensions: coherence, question diversity, coverage, and semantic accuracy, using a 1–5 scale. As shown in Table 2, the results confirm the dataset's effectiveness for training and evaluating dialogue systems. Additionally, we recalculated Cohen's Kappa and obtained a score of 85.76, indicating a high level of inter-rater agreement. More information on the manual evaluation can be found in Appendix **??**.

	train	dev	test
Coherence	4.48	4.31	4.17
Question Diversity	4.16	4.08	4.01
Semantic Accuracy	4.68	4.52	4.38

Table 2: Human evaluation results.

5.3 Comparison with Other Datasets

As shown in Table 4, the table compares several NL2GQL datasets, with MTGQL standing out as the only multi-turn dataset. Unlike other single-turn datasets, MTGQL is specifically designed to handle more complex, multi-turn queries, making it particularly suitable for tasks that require multiple interactions. Therefore, MTGQL will play a pivotal role in advancing research in multi-turn NL2GQL. For a more detailed description of the dataset generation methodology and dataset analysis, please refer to Appendix 9.4.

6 Models and Experimental Setup

6.1 Benchmark Methods

In-context learning with all schema method (ICL-AS): This method provides a set of examples within the input prompt, which concatenates all schema information and the question, guiding the LLM to generate the corresponding GQL.

Related schema extraction method (RSE): During training, this method uses the related schema and question as input, with the labeled GQL as output, while fine-tuning the LLM. In inference, it guides the LLM to extract related schema.

	train	dev	test	total
Number of Data Points	3000	500	1000	4500
Total Number of GQLs	19320	3252	6624	29196
Average Dialogue Turns per Data	6.44	6.50	6.62	6.49
Average entity per Data	4.64	4.89	5.17	4.79
Average relation per Data	5.47	5.65	5.93	5.59

Table 3: Basic Statistics of the Dataset.

Dataset	Language	Multi or Single	Domain	Syntax	Number
SpCQL (Guo et al., 2022)	Chinese	Single	Open-domain	Cypher	10000
CySpider (Zhao et al., 2023)	English	Single	Open-domain	Cypher	4929
Text2Cypher (Ozsoy et al., 2024)	English	Single	Open-domain	Cypher	44387
FinGQL (Liang et al., 2024a)	Chinese	Single	Finance	nGQL	-
MediGQL (Liang et al., 2024a)	Chinese	Single	Medicine	Cypher	-
R^3-NL2GQL (Zhou et al., 2024b)	Chinese English	Single	Open-domain	nGQL	-
StockGQL (Liang et al., 2024b)	Chinese	Single	Stock	nGQL	6456
TCMGQL (Liu et al., 2024)	Chinese	Single	Medicine	Cypher	-
EduGQL (Liu et al., 2024)	Chinese	Single	Education	Cypher	-
MTGQL(Ours)	Chinese	**Multi**	Stock	nGQL	4500

Table 4: A summary of the main NL2GQL datasets. From this, we can conclude that MTGQL is the only multi-turn dataset. The "-" in the Number column indicates that the dataset has not been open-sourced yet.

Fine-tuning with with all schema method (FT-AS): Approach concatenates all schema information with the question as input while applying LoRA for parameter-efficient fine-tuning of the base LLM.

Dependency-aware method (DA): We adapt the Dependency-aware Multi-turn Dataset Construction Framework with minor modifications and follow the method proposed in (Liang et al., 2024b) to construct a dependency-aware baseline. The adapted method comprises three key modules: a *Context Manager*, a *GQL Generator*, and a *GQL Refiner*. First, the **Context Manager** maintains the dialogue history, including previous questions, corresponding GQL queries and answers, as well as the involved entities and relations. It reformulates the current question based on the dialogue history to make it more formal and information-rich. Additionally, it extracts the relevant sub-schema for the current turn. Second, the **GQL Generator** generates a GQL query based on the reformulated question and the extracted sub-schema. Third, the **GQL Refiner** improves the generated query by refining it based on its execution results to enhance accuracy and relevance. More details are provided in Appendix 9.8.

6.2 Experimental Setup

Evaluation Metrics. The work in (Guo et al., 2022) introduced Exact Match (EM) and Exact Explanation (EX) for single-turn tasks. For multi-turn tasks, we propose Overall Exact Match (AEM) and Overall Exact Explanation (AEX), where all turns in a dialogue must be correct for the data to be considered successful. The formulas are as follows:

$$EM = \frac{\text{number of GQLs with a correct logical form}}{\text{total number of GQL}} \quad (1)$$

$$\text{AEM} = \frac{\text{number of data points with all GQLs having correct logical form}}{\text{total number of data points}} \quad (2)$$

$$EX = \frac{\text{number of GQLs with a correct execution result}}{\text{total number of GQL}} \quad (3)$$

$$\text{AEX} = \frac{\text{number of data points with all GQLs having correct execution results}}{\text{total number of data points}} \quad (4)$$

Implementation Details. Our experiments were conducted on an A800 GPU. We selected Qwen2.5-14B-Instruct (Team, 2024), LLaMA-3.1-8B-Instruct (Dubey et al., 2024), and GLM-4-9B-Chat (GLM et al., 2024) as the LLM backbone

Method	Backbones	EM(%)	AEM(%)	EX(%)	AEX(%)
ICL-AS	GLM-4-9B-Chat	31.13	6.50	30.01	5.80
	LLaMA-3.1-8B-Instruct	27.66	6.10	27.76	6.40
	Qwen2.5-14B-Instruct	32.55	7.50	29.70	7.20
	ChatGPT-4o	38.29	10.9	36.28	8.80
RES	GLM-4-9B-Chat	56.91	25.70	53.64	22.30
	LLaMA-3.1-8B-Instruct	58.76	27.10	56.63	26.70
	Qwen2.5-14B-Instruct	59.60	28.30	57.71	26.80
FT-AS	GLM-4-9B-Chat	60.14	30.60	56.16	28.80
	LLaMA-3.1-8B-Instruct	61.23	31.10	60.19	29.20
	Qwen2.5-14B-Instruct	63.56	31.50	61.70	31.20
DA	GLM-4-9B-Chat	65.53	38.70	63.47	36.60
	LLaMA-3.1-8B-Instruct	66.73	38.40	63.36	37.20
	Qwen2.5-14B-Instruct	**68.45**	**40.60**	**65.39**	**38.30**

Table 5: The comparison between the baseline methods is shown, with the bold numbers indicating the best results.

models. In this paper, all sequence encoding is performed using the all-MiniLM-L6-v2 model, with the embedding dimension set to 384. All the number of demonstrations K are set as 4.

7 Results

7.1 Main Results

Based on the results presented in Table 5, the DA method consistently outperforms all other approaches across all evaluation metrics. Notably, when combined with the Qwen2.5-14B-Instruct backbone, DA achieves the highest scores in EM (**68.45%**), AEM (**40.60%**), EX (**65.39%**), and AEX (**38.30%**). In contrast, the ICL-AS method yields comparatively lower results, which can be attributed to the absence of high-quality GQL-related corpora during the pretraining of its underlying models. Moreover, performance differences observed across various backbone models within the same method underscore the substantial impact of model architecture and backbone selection on the final outcomes. This highlights the necessity of carefully choosing and aligning the model backbone with the specific demands of the task. Nevertheless, it is worth noting that the overall accuracy on this task remains relatively low, suggesting that there is still considerable room for improvement.

7.2 Breakdown Results by Round

Table 6 presents the results of the best baseline method across different rounds, showing a clear decline in performance as rounds increase. This decrease is likely due to the increasing complexity of multi-turn interactions, which challenges the model's ability to maintain context and generate consistent responses.

Round	EM(%)	EX(%)
R1	84.21	82.88
R2	73.66	73.13
R3	60.25	58.44
R4	47.84	46.18
R5+	31.23	30.96

Table 6: The breakdown of results by round, where R1-R4 represent rounds 1 to 4, and R5+ denotes round 5 and beyond.

Round	EM(%)	EX(%)
P1	70.47	68.49
P2	64.70	63.66
P3	66.52	64.12
P4	73.84	71.68
P5	62.59	62.32
P6	67.36	66.46

Table 7: Results by the question expansion pattern.

Table 7 shows performance across different question expansion patterns, with notable variations. These fluctuations indicate that the model is more effective with simpler question expansions (like P1 and P4), while more complex patterns (like P2 and P5) lead to lower accuracy, likely due to the increased difficulty of generating precise answers. More experimental analyses are provided in Appendix 9.9.

8 Conclusion

In this paper, we introduce a dependency-aware multi-turn dataset construction framework for building multi-turn NL2GQL datasets. Using this framework, we create MTGQL, the first multi-turn NL2GQL dataset. Finally, we propose three baseline methods based on this dataset, laying the groundwork for future advancements in the field.

Limitations

There are several limitations that we would like to address in future work.

First, although we have developed a Chinese multi-turn NL2GQL dataset, we have not yet completed the translation into English due to the extensive amount of entity and relation names that require translation from the graph database. Once this process is completed, we plan to release a bilingual (Chinese-English) version of the dataset as open source to facilitate broader research adoption.

Second, while our dataset supports multi-turn queries involving complex contextual dependencies, the current benchmark methods rely on manually designed schemas or dependency-aware modules. These methods may not generalize well to unseen domains or schema structures. Future work could explore schema-agnostic approaches or large-scale pretraining on multi-turn graph querying tasks.

Third, the current evaluation focuses primarily on execution accuracy of generated GQL. However, execution correctness may not fully capture semantic correctness or partial matching of subgraph intents. Incorporating human evaluation or developing more fine-grained metrics could provide better insights into model behavior.

Lastly, although our dataset construction process includes context reformulation and sub-schema extraction, the pipeline still involves certain heuristic rules and prompt designs that may not scale well across diverse graph domains. We aim to further automate and generalize the dataset construction framework to reduce reliance on manual tuning.

References

Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751*.

Bosheng Ding, Chengwei Qin, Ruochen Zhao, Tianze Luo, Xinze Li, Guizhen Chen, Wenhan Xia, Junjie Hu, Anh Tuan Luu, and Shafiq Joty. 2024. Data augmentation using large language models: Data perspectives, learning paradigms and challenges. *Preprint*, arXiv:2403.02990.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

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*, CIKM '22, page 3973–3977, New York, NY, USA. Association for Computing Machinery.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Yuanyuan Liang, Keren Tan, Tingyu Xie, Wenbiao Tao, Siyuan Wang, Yunshi Lan, and Weining Qian. 2024a. Aligning large language models to a domain-specific graph database for nl2gql. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1367–1377.

Yuanyuan Liang, Tingyu Xie, Gan Peng, Zihao Huang, Yunshi Lan, and Weining Qian. 2024b. Nat-nl2gql: A novel multi-agent framework for translating natural language to graph query language. *arXiv preprint arXiv:2412.10434*.

Yang Liu, Xin Wang, Jiake Ge, Hui Wang, Dawei Xu, and Yongzhe Jia. 2024. Text to graph query using filter condition attributes. *Proceedings of the VLDB Endowment. ISSN*, 2150:8097.

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On LLMs-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11065–11082, Bangkok, Thailand. Association for Computational Linguistics.

André Lopes, Diogo Rodrigues, João Saraiva, Maryam Abbasi, Pedro Martins, and Cristina Wanzeller. 2023. Scalability and performance evaluation of graph database systems: A comparative study of neo4j, janusgraph, memgraph, nebulagraph, and tigergraph. In *2023 Second International Conference On Smart*

Technologies For Smart Nation (SmartTechCon), pages 537–542. IEEE.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2024. Text2cypher: Bridging natural language and graph databases. *arXiv preprint arXiv:2412.10064*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*.

Robert Pavliš. 2024. Graph databases: An alternative to relational databases in an interconnected big data environment. In *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, pages 247–252. IEEE.

Yongduo Sui, Qitian Wu, Jiancan Wu, Qing Cui, Longfei Li, Jun Zhou, Xiang Wang, and Xiangnan He. 2024. Unleashing the power of graph data augmentation on covariate distribution shift. *Advances in Neural Information Processing Systems*, 36.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Wenbiao Tao, Hanlun Zhu, Keren Tan, Jiani Wang, Yuanyuan Liang, Huihui Jiang, Pengcheng Yuan, and Yunshi Lan. 2024. Finqa: A training-free dynamic knowledge graph question answering system in finance with llm-based revision. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 418–423. Springer.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Quoc-Bao-Huy Tran, Aagha Abdul Waheed, and Sun-Tae Chung. 2024. Robust text-to-cypher using combination of bert, graphsage, and transformer (cobgt) model. *Applied Sciences*, 14(17):7881.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Ran Wang, Zhengyi Yang, Wenjie Zhang, and Xuemin Lin. 2020. An empirical study on recent graph database systems. In *Knowledge Science, Engineering and Management: 13th International Conference, KSEM 2020, Hangzhou, China, August 28–30, 2020, Proceedings, Part I 13*, pages 328–340. Springer.

Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *Preprint*, arXiv:2402.18013.

Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. 2022a. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*.

Ziyu Zhao, Wei Liu, Tim French, and Michael Stewart. 2023. Cyspider: A neural semantic parsing corpus with baseline models for property graphs. In *Australasian Joint Conference on Artificial Intelligence*, pages 120–132. Springer.

Ziyu Zhao, Michael Stewart, Wei Liu, Tim French, and Melinda Hodkiewicz. 2022b. Natural language query for technical knowledge graph navigation. In *Australasian Conference on Data Mining*, pages 176–191. Springer.

Yue Zhou, Chenlu Guo, Xu Wang, Yi Chang, and Yuan Wu. 2024a. A survey on data augmentation in large model era. *Preprint*, arXiv:2401.15422.

Yuhang Zhou, Yu He, Siyu Tian, Yuchen Ni, Zhangyue Yin, Xiang Liu, Chuanjun Ji, Sen Liu, Xipeng Qiu, Guangnan Ye, and Hongfeng Chai. 2024b. r^3-NL2GQL: A model coordination and knowledge graph alignment approach for NL2GQL. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13679–13692, Miami, Florida, USA. Association for Computational Linguistics.

9 Appendix

9.1 Comparison with Similar Tasks

Text2SQL

While numerous highly effective Text2SQL methods have been developed (Caferoğlu and Ulusoy, 2024; Wang et al., 2023; Talaei et al., 2024), the fundamental differences between GQL and SQL present significant challenges for directly applying these methods to the NL2GQL task. Several studies have examined the differences between Text2SQL and NL2GQL (Guo et al., 2022; Liang et al., 2024a; Zhou et al., 2024b), and we highlight the key distinctions in the following areas:

- **Differences in standard syntax:** Unlike SQL, which follows a standardized query language, GQL lacks a unified standard. Different graph databases adopt distinct query languages such as Cypher, nGQL, and Gremlin. This fragmentation complicates dataset construction, model generalization, and the development of consistent training paradigms.
- **Differences in query types:** GQL surpasses the typical CRUD operations by offering advanced query types like sub-graph and path queries that enable complex data traversal. Its extensive keyword set further enhances its flexibility, making it a powerful tool for a wide range of data manipulation needs.
- **Differences in translation difficulties:** NL2GQL involves understanding graph structures, path reasoning, and pattern matching, requiring high query flexibility, which may lead to issues such as path combination explosion. In contrast, Text-to-SQL faces challenges like pattern matching, table/column name mapping, and SQL syntax parsing, but the overall query structure remains relatively stable.
- **Differences in language model capabilities:** Text-to-SQL benefits from a large corpus and extensive datasets, while NL2GQL has far fewer resources. Given that most widely used pre-trained models, especially LLMs, rely on pre-training followed by fine-tuning, this disparity in resources directly impacts their performance on these tasks.

In conclusion, due to the substantial differences between the two, it is essential to develop specialized approaches for NL2GQL rather than simply adapting Text-to-SQL methods.

Multi-turn Dialogue

Multi-turn dialogue systems involve an iterative, back-and-forth exchange between a user and a system, where the conversation evolves over multiple turns. These systems aim to refine user queries, explore topics in more depth, and generate contextually appropriate responses based on previous interactions. Unlike single-turn dialogue systems, which address isolated queries, multi-turn dialogues manage dynamic and context-sensitive information flows (Yi et al., 2024).

Multi-turn NL2GQL is a specialized form of Multi-turn Dialogue. Unlike other Multi-turn Dialogue systems, NL2GQL focuses on converting natural language into GQL based on a graph database. This distinction makes Multi-turn NL2GQL ideal for dynamic interactions with graph-based data, where each query may involve traversing different paths or nodes. The model must not only understand the current query but also retain information from previous interactions to generate accurate, contextually relevant graph queries. This ability to maintain coherence across multiple turns poses challenges in handling complex graph traversals and evolving contexts.

Multi-turn Knowledge Base Question Answering. A knowledge graph is a structured knowledge base represented as a graph, designed to organize vast amounts of real-world information in a flexible and scalable manner. Its primary goal is to enable machines to understand this information and perform reasoning and inference (Zhao et al., 2022b; Pan et al., 2024). In contrast, a graph database primarily focuses on efficient data storage and query optimization, rather than on knowledge reasoning and semantic understanding. As such, KBQA emphasizes knowledge-based reasoning and semantic understanding to extract answers from structured knowledge bases, while NL2GQL focuses on constructing effective graph queries.

A typical example of a problem that NL2GQL can solve but KBQA cannot is as follows:

Problem: Find all users who participated in at least two projects in 2023, and whose friends include at least one person from the R&D department.

NL2GQL Solution: The complex graph traversal logic can be directly expressed using graph query languages like Cypher Pseudo-code:

```
MATCH (u:User)-[:PARTICIPATED_IN]->(
p:Project {year: 2023})
WITH u, COUNT(p) AS project_count
WHERE project_count >= 2
MATCH (u)-[:FRIEND_OF]->(f:User)-
[:BELONGS_TO]->(:Dept {name: "R&D"})
RETURN u.name, COLLECT(f.name)
       AS friends_in_rd
```

Why KBQA Struggles with This Problem:

- **Multi-hop Relationship Traversal:** This problem requires reasoning across 4 hops: User → Project → Count → Friend → Department. Traditional KBQA systems typically handle only single-hop or fixed-path queries and are not equipped to flexibly manage dynamic path lengths (e.g., recursive traversal of the "FRIEND_OF" relationship).

- **Aggregation and Conditional Combination:** The task involves both an aggregation operation (e.g., COUNT(p) >= 2) and a conditional filter (e.g., friends from the R&D department). KBQA systems usually cannot combine aggregation functions with multiple entity conditions within the same query.

- **Implicit Logical Dependencies:** The condition "at least one friend belongs to the R&D department" necessitates an existence check (EXISTS) rather than a simple attribute match. KBQA typically returns explicitly stored triples and cannot dynamically infer such existence conditions.

Other NL2GQL-exclusive Capabilities include the following question examples:

- **Path Queries:** Question: "Find the shortest collaboration path from User A to User B, where all nodes in the path are employees who joined after 2020."

  Cypher Pseudo-code:

  ```
  MATCH (a:User {name: "UserA"}),
   (b:User {name: "UserB"}),
   path = shortestPath((a)-
   [:COLLABORATES_WITH*]-(b))
  WHERE ALL(node IN nodes(path)
  WHERE node:Employee AND
   node.join_date >= '2020-01-01')
  RETURN path
  ```

- **Dynamic Pattern Reasoning:** Question: "Count the managers in all departments who have more than 10 subordinates and whose subordinates have participated in cross-departmental projects."

  Cypher Pseudo-code:

  ```
  MATCH (dept:Department)
   <-[:MANAGES]-(manager:Manager)
  WITH dept, manager, [(manager)-
   [:MANAGES]->(emp:Employee) | emp]
   AS subordinates
  WHERE size(subordinates) > 10
   AND ANY(emp IN subordinates
  WHERE EXISTS {
      MATCH (emp)-[:PARTICIPATED_IN]
       ->(proj:Project)
      WHERE proj.department
       <> dept.name
      })
  RETURN dept.name AS department,
   manager.name AS manager,
   size(subordinates) AS emp_count
  ```

- **Temporal Graph Analysis:** Question: "List all stocks that experienced a drop of more than 5% in a single day after 5 consecutive days of price increases."

  Cypher Pseudo-code:

  ```
  MATCH (s:Stock)-[r:HAS_DAILY_DATA]
      ->(d:DailyData)
  WITH s, d ORDER BY d.date ASC
  WITH s, COLLECT(d) AS data
  WHERE size(data) >= 6
    AND ANY(i IN RANGE(0,
      size(data)-6)
    WHERE
      REDUCE(isRising = true,
       j IN [0..4] |
       isRising AND
       data[i+j+1].close_price >
        data[i+j].close_price
      )
      AND (data[i+5].close_price -
       data[i+6].close_price) /
       data[i+5].close_price >= 0.05
  RETURN s.name AS stock,
   data[i+5].date AS peak_date,
   data[i+6].date AS drop_date
  ```

Algorithm 1: Question Expansion Pattern Selection Algorithm

Input: Set of entities and relations $\{E, R\}$, schema of G, set of expansion patterns $\{P1, P2, P3, P4, P5, P6\}$

Output: Selected expansion pattern and corresponding entities and relations

Step 1: Expansion Pattern Filtering
for *each expansion pattern* P_i *in* $\{P1, P2, \ldots, P6\}$ **do**
    if *Pattern* P_i *meets the predefined conditions based on* E*,* R*, and* G **then**
        Include P_i in the set of valid patterns
    else
        Remove P_i from the set of valid patterns
Step 2: Expansion Pattern Selection
for *each valid expansion pattern* P_i **do**
    Set initial weight of P_i as $w(P_i) = \frac{1}{6}$
for *each previously used expansion pattern* P_i **do**
    Halve its weight: $w(P_i) = \frac{w(P_i)}{2}$
    Redistribute the halved weight equally among other remaining patterns
Select the expansion pattern P_{selected} with the highest weight:
$P_{\text{selected}} = \arg\max_{P_i} w(P_i)$
Step 3: Entity and Relation Selection
Determine the potential candidate entities $E_{\text{candidates}}$ based on P_{selected}
for *each candidate entity* $e \in E_{candidates}$ **do**
    Set initial weight of entity e as $w(e) = \frac{1}{|E_{\text{candidates}}|}$
    if e *has been referenced in the previous dialogue step* **then**
        Increase $w(e)$ by $\frac{1}{4}$, indicating higher likelihood of selection
    Redistribute the increased weight evenly among other remaining entities
Determine the potential relations $R_{\text{candidates}}$ based on P_{selected}
for *each relation* $r \in R_{candidates}$ **do**
    Assign weight to r using a similar process as entity selection
return *Selected expansion pattern* $P_{selected}$*, selected entities, and selected relations*

9.2 Question expansion patterns selection algorithm.

In this section, we present our question expansion pattern selection algorithm, a key innovation of this work. As described in Section 4.2, the Context Manager stores a set of entities and relations, along with six expansion patterns.

As illustrated in Algorithm 1, our algorithm follows three main steps:

- **Expansion Pattern Filtering:** Based on the set of entities, relations, and the schema of G, we sequentially evaluate the conditions for each of the six expansion patterns (P1-P6) using predefined rules. We filter out the patterns that do not meet the necessary conditions.

- **Expansion Pattern Selection:** From the remaining expansion patterns, we select the most appropriate one according to their assigned weights. Initially, each pattern is given a weight of 1/6. If a pattern has already been used, its weight is halved, and the reduced weight is evenly distributed among the other remaining patterns.

- **Entity and Relation Selection:** Once the expansion pattern is selected, we proceed to choose the corresponding entities and relations. In the entity selection process, we first identify the potential candidate entities based on the chosen pattern. Then, we assign weights to these entities. Initially, each potential entity receives an equal weight of 1/|E|, where |E| is the total number of candidate entities. If an entity has been refer-

enced in the previous step of the dialogue, its weight increases by 1/4, indicating a higher likelihood of its selection in the current step. The increased weight is evenly redistributed among the remaining entities to maintain balance. The relation selection follows a similar approach.

9.3 Prompt for Question Generation

As shown in Figure 3, this prompt generates clear and contextually relevant questions based on a schema and dialogue history, following a question expansion pattern. It guides the LLM to produce either an opening question or a follow-up question that incorporates colloquial, informal, and ambiguous expressions to better simulate real user queries, using entity placeholders according to the expansion pattern. The output includes both a raw question with references and a fully disambiguated version, free of placeholders and references, ensuring contextual relevance and structural clarity. It is worth noting that, since we are constructing a Chinese dataset, the prompt is originally written in Chinese. For ease of reading, however, we have provided an English translation.

9.4 Analysis of Dataset Generation Methodology and Dataset Characteristics

9.4.1 Detailed Mechanisms of Dataset Construction Components

In order to explain more detailed descriptions of the internal mechanisms of our dataset construction framework components, we provide the following explanations for the key modules: Question Generator, GQL Generator, and GQL Validator and Optimizer.

Question Generator. The Question Generator leverages a LLM to produce contextually coherent questions by conditioning on the dialogue history and relevant schema information. Specifically, the LLM is prompted with both previous turns in the conversation and masked templates to ensure that the generated questions maintain semantic continuity and relevance to the evolving dialogue context. Detailed prompt designs and example outputs are provided in Figure 3.

GQL Generator. To convert natural language questions into executable GQL commands, the GQL Generator employs a fine-tuned LLM guided by the complete database schema. The generator incorporates the full schema context and uses the reformulated question, which includes disambiguated references and expanded context, to produce accurate and context-aware GQL queries. This approach is inspired by the method described in (Liang et al., 2024b), which effectively integrates schema constraints to generate GQL.

GQL Validator and Optimizer. The GQL Validator and Optimizer modules are responsible for the semantic verification and refinement of generated queries. The Validator executes the generated GQL query against the graph database and compares the results with the expected outcomes inferred from the dialogue context to identify any discrepancies. Upon detecting inconsistencies, the Optimizer uses carefully designed prompts—identical to the refiner prompts described in (Liang et al., 2024b)—to guide the LLM in iteratively revising and improving the query. These prompts emphasize error correction, adherence to the database schema, and maintaining contextual consistency. Further details regarding the prompt design and the iterative optimization process can be found in lines 355–368 of this paper.

Together, these components form a tightly integrated framework that ensures generated questions and GQL queries are both contextually coherent and semantically accurate, thereby effectively supporting the construction of a high-quality multi-turn NL2GQL dataset.

9.4.2 Effectiveness of Dataset-Based Training for GQL Generation

The core question raised concerns the ability of current LLMs to generate high-quality multi-turn GQL dialogues, particularly in the absence of task-specific training data. While LLMs such as ChatGPT or Qwen2.5 can generate GQL queries without fine-tuning, the accuracy of such outputs is far from guaranteed. Our framework incorporates a dataset-driven training process to enhance the precision of generated queries and reduce the loss of usable data due to filtering invalid outputs. To date, there exists no more effective method for reliably improving GQL generation quality, especially in complex multi-turn scenarios.

To better understand the effectiveness of our training method and the necessity of filtering, we conducted two additional evaluations:

- **(1) Direct generation without filtering:** We generated 1,000 multi-turn dialogue samples without applying any error filtering or training.

Instruction:
You are an expert in both language processing and NebulaGraph. Given the schema, question expansion pattern, and dialogue history, generate a clear, relevant, and contextually appropriate question by following the rules below:

1. Generate a question based on the schema and dialogue context, ensuring it is contextually relevant and logically continues the conversation. The question should be conversational in style, incorporating ellipses, omissions, and vague expressions wherever appropriate.
2. Use placeholders for entities, such as: `[s]` for stock, `[c]` for chairman, `[h]` for stockholder, `[t]` for trade, `[p]` for public offering fund, `[f]` for fund manager, `[i]` for industry, `[d]` for time, and `[m]` for numbers.
3. If the dialogue history is empty, create an opening question. If there is existing dialogue, generate a follow-up question that aligns with the provided question expansion pattern.
4. Generate the raw question in a conversational style, incorporating relevant references.
5. Generate the formal question based on the raw question. The formal question should be a disambiguated version of the raw question, clarified and free of placeholders or references.

Input:
1. Schema Information:
{SCHEMA}
2. Dialogue History:
{DIALOGUE_HISTORY}
3. Question Expansion Pattern:
{QUESTION_EXPANDING_PATTERN}

Output:
Provide the generated raw question after "Question" and the formal question after "Complete Question" directly.

Question:

Complete Question:

Figure 3: The prompt for question generation.

The results show that the execution accuracy (EX) for single-turn queries was only **39.8%**, while the overall multi-turn accuracy (AEX) dropped to just **8.4%**. This highlights the poor reliability of direct generation without task-specific fine-tuning or filtering mechanisms

- **(2) Fine-tuning with limited data:** We fine-tuned the GQL generator using only 500 annotated samples under the "fine-tuning with all schema" setting and evaluated it on the same benchmark test set as in our main experiments. The resulting execution accuracy (EX) and average execution accuracy (AEX) were **29.99%** and **15.42%**, respectively—substantially lower than the best-performing results reported in our main paper (EX: **65.39%**, AEX: **38.30%**). These results further confirm the importance of using a high-quality, sufficiently large training set for accurate GQL generation in multi-turn settings.

Moreover, Table 6 reveals a dramatic **50%** performance drop in both EM and EX scores from Round 1 (R1) to Rounds 5+ (R5+), highlighting that the primary bottleneck lies in maintaining contextual understanding and reasoning across multiple dialogue turns, rather than in single-turn query generation.

These findings suggest that the key limitation is not the dataset itself but rather the inherent difficulty of maintaining dialogue coherence and rea-

soning across multiple conversational turns. Consequently, targeted dataset design and fine-tuning remain critical components in improving multi-turn GQL generation.

It is worth reiterating that directly using LLMs to generate GQL queries often results in low accuracy, far from being satisfactory for practical use. This necessitates a post-processing pipeline that filters and optimizes the generated GQLs. Our primary goal is to construct a high-quality multi-turn NL2GQL dataset, where maintaining the coherence and scalability of natural language questions is crucial. Given the initially low quality of GQLs produced by the LLM, we apply strict filtering to remove a large portion of erroneous intermediate outputs, thereby ensuring the reliability of the final dataset.

Furthermore, as shown in Table 5, the LLM fine-tuned on our generated dataset significantly outperforms the ICL-based approach across multiple evaluation metrics. This demonstrates that our dataset effectively enhances the LLM's ability to understand and generate accurate graph queries in multi-turn scenarios.

9.4.3 Handling of Historical Information in Multi-turn NL2GQL

In our MTGQL dataset and baseline methods, we explicitly model the interdependency of dialogue history to handle multi-turn queries. Specifically, rather than simply concatenating the entire dialogue sequence, we employ a structured approach in which the dialogue context consists of:

- **Previous questions** — to provide linguistic and semantic context;
- **Previously generated GQL queries** — to preserve formal query structures and constraints;
- **Execution results or answers of prior queries** — to help verify correctness and guide refinements;
- **Entities and relations involved in prior turns** — to focus on relevant schema components.

This structured context is maintained and managed by the *Context Manager* module (described in Section 4.2), which reformulates the current user question into a more explicit and self-contained query by referencing the above components. This reformulated question, together with an extracted relevant sub-schema, is then passed to the GQL generation and refinement modules.

We use prompt templates that incorporate these historical elements to guide the language model in generating accurate and context-aware GQL statements. This approach goes beyond naive sequence concatenation by leveraging execution feedback and schema relevance, improving the handling of coreferences, ellipsis, and multi-turn dependencies.

9.4.4 Keyword Analysis

To evaluate the richness and syntactic diversity of query expressions in the StockGQL dataset, we conducted a keyword frequency analysis across the training, development, and test sets. Specifically, we focused on core nGQL-related terms, categorized as follows:

- **Query Control:** `MATCH`, `GO`, `FETCH`, `LOOKUP`, `WHERE`, `YIELD`, `WITH`, `LIMIT`, `ORDER BY`, `GROUP BY`, `RETURN`
- **Logical Operators:** `AND`, `OR`, `NOT`, `XOR`
- **Graph Traversal:** `VERTEX`, `EDGE`, `OVER`, `REVERSELY`, `BIDIRECT`
- **Aggregation Functions:** `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`, `COLLECT`, `DISTINCT`

Excluding structural keywords such as `MATCH` and `RETURN`, which appear in nearly all queries by default, the results in Table 8 show that each dataset split contains a substantial number of informative and diverse keywords. Notably, the test set contains an average of more than 2.1 such keywords per sample. This reflects the high syntactic complexity and operational diversity of StockGQL, highlighting its effectiveness as a benchmark for evaluating the expressive capabilities of NL2GQL models.

	Total Keywords	GQL Count	Avg
Train	20479	19320	1.06
Dev	3448	3252	1.16
Test	7352	6624	1.11

Table 8: Statistics of nGQL keyword usage in the StockGQL dataset.

9.4.5 Query Type Statistics

To better understand the distribution of query intents in the MTGQL dataset, we following the question type categorization framework proposed in (Liang et al., 2024a), we conducted a comprehensive statistical analysis of StockGQL. As shown in Table 9, StockGQL covers a diverse range of query types, with particularly high representation in complex categories such as Numerical Sorting, Relationship Filtering, and Relationship Inference.

	train	dev	test
Entity property	2145	345	770
Numerical sorting	4039	841	1448
Relationship inference	2585	415	891
Yes/No question	1281	249	473
Relationship filtering	4276	602	1396
Attribute comparison	1897	274	782
Edge property	1923	272	635
String filtering	1174	254	229

Table 9: Performance of our method on various types of queries in the FinGQL dataset.

9.5 Expansion Patterns and Alignment with User Behavior

While it is inherently challenging to ensure that automatically generated questions fully capture the diversity of real user behavior, our goal is to approximate realistic multi-turn interaction scenarios as closely as possible.

To this end, we define six expansion patterns, each designed to reflect common user intents—such as refining a previous query, shifting focus to related entities, or requesting aggregated information. As shown in Table 1, these patterns offer structural guidance during data generation. We also include representative examples to illustrate how each pattern constrains and informs the generation of follow-up questions in a multi-turn setting.

Furthermore, as demonstrated in Prompt 3, these patterns are explicitly embedded in the prompt instructions provided to the LLM. We additionally require that the generated questions adopt a conversational tone, featuring ellipses, omissions, and vague expressions where appropriate. These expansion patterns act as soft constraints that help the LLM maintain coherence, contextual relevance, and logical progression across dialogue turns, thereby improving the plausibility and utility of the resulting dataset.

9.6 Generalization to Other Datasets

To evaluate the generalization capability of our proposed multi-turn dataset **MTGQL**, we conducted cross-dataset transfer experiments on **StockGQL**(Liang et al., 2024b). Specifically, following the method in(Liang et al., 2024b), we fine-tuned the same GQL generator model on MTGQL and directly evaluated it on the StockGQL test set, without any further fine-tuning on StockGQL data.

The results are summarized in Table 10. We observe that, although the model trained solely on MTGQL does not surpass models directly trained on StockGQL, it still achieves competitive performance, with EM and EX scores exceeding 80

Additionally, we explored a joint training strategy where the model was first trained on MTGQL and then fine-tuned on StockGQL. This setting yielded consistent improvements of approximately 4.5% across all metrics compared to training on StockGQL alone. These results suggest that MTGQL serves as a valuable complementary resource, enhancing the generalization ability and robustness of models for NL2GQL tasks.

Training Dataset	**EM (%)**	**EX (%)**
StockGQL only	85.44	86.25
MTGQL only	81.61	80.23
MTGQL + StockGQL	**90.15**	**90.89**

Table 10: Cross-dataset evaluation: training on MTGQL and testing on StockGQL.

9.7 Manual Evaluation Protocol

To assess the dataset's quality, we conducted a human evaluation involving three domain experts. They independently rated 200 randomly sampled dialogues from each split (train, dev, test), totaling 600 dialogues. Each dialogue was evaluated on four dimensions:

- **Coherence**: logical flow across dialogue turns.
- **Question Diversity**: variety in question types and forms.
- **Coverage**: breadth of entities and relations involved.
- **Semantic Accuracy**: alignment of questions with the schema and their meaningfulness.

Each dimension was scored on a 1–5 scale, where 1 = *very poor*, 2 = *poor*, 3 = *fair*, 4 = *good*, and 5 = *excellent*. Detailed guidelines for the scoring are as follows:

- **Coherence**:
  - 1: Dialogue is incoherent or inconsistent.
  - 2: Frequent logical gaps.
  - 3: Partially coherent with some abrupt transitions.
  - 4: Mostly logical and connected.
  - 5: Fully coherent and natural dialogue flow.
- **Question Diversity**:
  - 1: Highly repetitive questions.
  - 2: Limited variation in question form or content.
  - 3: Moderate diversity.
  - 4: Good variation in question types.
  - 5: Broad and rich variety of question forms and intents.
- **Coverage**:
  - 1: Very narrow focus on one topic or entity.
  - 2: Minor variation in entities or relations.
  - 3: Involves a few distinct schema elements.
  - 4: Covers a range of entity and relation types.
  - 5: Broad and comprehensive schema coverage.
- **Semantic Accuracy**:
  - 1: Questions are semantically invalid or nonsensical.
  - 2: Multiple inconsistencies with schema.
  - 3: Generally valid but with minor semantic flaws.
  - 4: Mostly correct and meaningful.
  - 5: Fully accurate, meaningful, and well-grounded in the schema.

Each dialogue was evaluated independently by all three experts, and the final score per dimension was averaged. To ensure consistency in annotation, we computed inter-rater agreement using Cohen's Kappa, which yielded a score of **85.76**, indicating a high level of annotation reliability.

9.7.1 Example Case Analysis

Here is a sample dialogue excerpt and its evaluation:

- **Dialogue:**
  - Q1: "Who is the CEO of `[Company A]`?"
  - Q2: "What subsidiaries does it own?"
  - Q3: "Among them, which were founded after 2010?"
- **Expert Scores:**
  - **Coherence: 5** — Each turn builds naturally on the previous.
  - **Diversity: 4** — Mix of factoid and temporal questions.
  - **Coverage: 5** — Involves various entity types (company, person, subsidiary, time).
  - **Semantic Accuracy: 5** — All questions align well with the schema and are meaningful.

9.8 Dependency-aware Method

To address the challenge of modeling multi-turn dependencies in NL2GQL, we propose a **Dependency-aware Method (DA)**, which extends the Dependency-aware Multi-turn Dataset Construction Framework with necessary adaptations, following the approach of (Liang et al., 2024b) and tailoring it to the MTGQL dataset setting.

The proposed DA method comprises three key components: a *Context Manager*, a *GQL Generator*, and a *GQL Refiner*. These components are designed to collaboratively maintain dialogue coherence, support context-sensitive reasoning, and generate accurate graph queries in multi-turn interactions. The pseudocode of the algorithm is shown in Algorithm 2.

Context Manager. This module is responsible for maintaining and organizing the dialogue history across turns. For each turn, it constructs a structured context that includes:

- Natural language questions from previous turns;
- Corresponding GQL queries generated in earlier turns;
- Execution results of those queries;

Algorithm 2: Dependency-aware Multi-turn NL2GQL Inference

Input: Graph database G; multi-turn dialogue $C = \{(Q_1, A_1), \ldots, (Q_{t-1}, A_{t-1})\}$; current question Q_t

Output: Executable GQL query GQL_t

Initialize: Structured context $\mathcal{H} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $t-1$ **do**
  Extract (Q_i, A_i) from C;
  $GQL_i \leftarrow$ previously generated query for Q_i;
  $Entities_i, Relations_i \leftarrow \texttt{Analyze}(GQL_i, A_i)$;
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{Q_i, GQL_i, A_i, Entities_i, Relations_i\}$;
$Q_t^{\text{explicit}} \leftarrow \texttt{Reformulate}(Q_t, \mathcal{H})$; // Resolve coreference and ellipsis
$SubSchema_t \leftarrow \texttt{ExtractRelevantSubSchema}(G, \mathcal{H}, Q_t^{\text{explicit}})$;
$GQL_t^{\text{init}} \leftarrow \texttt{GQLGenerator}(Q_t^{\text{explicit}}, SubSchema_t)$;
$A_t^{\text{pred}} \leftarrow \texttt{Execute}(GQL_t^{\text{init}}, G)$;
if $\texttt{IsAligned}(A_t^{pred}, Q_t, \mathcal{H})$ **then**
  $GQL_t \leftarrow GQL_t^{\text{init}}$;
else
  $GQL_t \leftarrow \texttt{Refine}(GQL_t^{\text{init}}, A_t^{\text{pred}}, Q_t, \mathcal{H})$;
return GQL_t

- Involved entities and relations, representing the dynamic subgraph explored so far.

Before generating the current turn's query, the Context Manager reformulates the user question into a more explicit, context-independent version. This includes resolving coreferences (e.g., "their", "its") and filling in ellipses. It also retrieves a relevant sub-schema by identifying schema elements mentioned in both the dialogue history and the current turn, ensuring precise grounding.

GQL Generator. Given the reformulated question and the retrieved sub-schema, this module utilizes a fine-tuned large language model (LLM) to generate a candidate GQL query. Following the method described in (Liang et al., 2024b), the generator aims to produce structurally and semantically accurate queries aligned with the user's intent in the current dialogue context.

GQL Refiner. Due to the inherent difficulty of GQL generation in complex multi-turn settings, we introduce a post-generation refinement step. The Refiner evaluates whether the generated query aligns with the intended meaning of the user input by analyzing its execution result. If inconsistencies are detected, the Refiner prompts the model to revise the query, improving execution correctness and robustness.

Collaboration Mechanism. The three components operate in a tightly coupled workflow. The Context Manager ensures that rich contextual information is provided to the GQL Generator, enabling it to account for prior dialogue turns. The GQL Generator then produces an initial query candidate, which is further validated and refined by the GQL Refiner. This collaborative mechanism ensures continuity, contextual fidelity, and high-quality query generation throughout the multi-turn process.

Overall, this dependency-aware pipeline bridges the gap between natural conversational flow and the generation of accurate, executable graph queries, thereby enabling robust and interpretable NL2GQL performance in complex multi-turn scenarios.

9.9 Further Experimental Results

9.9.1 Error Analysis

To better understand the limitations of our proposed baseline methods on the MTGQL dataset, we conduct a detailed error analysis across the four benchmark baselines: ICL-AS, RSE, FT-AS, and DA. We manually analyze 300 error cases sampled from the test set, categorizing them into distinct failure types inspired by prior analyses in Spider 2.0 (Lei et al., 2024) and adapted to the multi-turn NL2GQL setting.

Error Type	ICL-AS	RSE	FT-AS	DA
Schema Selection Errors	29%	25%	27%	18%
Contextual Understanding Failures	37%	28%	34%	21%
Logical Form Generation Errors	14%	22%	19%	13%
Ambiguity / Underspecification	13%	15%	12%	12%
Execution-based Errors	7%	10%	8%	6%

Table 11: Distribution of error types among different baseline methods on 300 sampled error cases.

Turn(s) and Prediction	Details and Error Type
Turn 1: *Show me the companies invested by Baidu.* **Turn 2:** *What about their subsidiaries?* **Prediction (ICL-AS):** Returns subsidiaries of all companies.	Fails to resolve "their" as referring to companies invested by Baidu. Contextual history is not retained, leading to incorrect scope. **Error Type:** Contextual Understanding Failure.
Turn: *Which listed companies are controlled by Tencent and operate in the finance sector?* **Prediction (RSE):** Omits "listed" constraint.	Schema extraction covers "Tencent" and "finance sector", but "listed" is ignored in generation due to weak schema grounding. **Error Type:** Logical Form Generation Error.
Turn: *How about its most recent investment?* **Prediction (FT-AS):** Returns any investment without ordering.	Fails to interpret "most recent" as temporal ordering. Lacks temporal reasoning or clarification strategy. **Error Type:** Ambiguity / Underspecification.

Table 12: Representative errors and analysis on MTGQL dataset.

1. Schema Selection Errors (26%) These errors arise when the model selects incorrect or incomplete schema elements (i.e., node or edge types) for the current turn. This is especially problematic in ICL-AS and FT-AS, which must reason over the entire schema without contextual focus. In multi-turn scenarios, the lack of dynamic schema narrowing often causes confusion, especially when the current utterance implicitly refers to earlier entities.

2. Contextual Understanding Failures (32%) These include failures where the model misunderstands the dependencies between the current utterance and the previous turns. For instance, co-reference resolution (e.g., "What about its subsidiaries?") or omitted subject/object references lead to incorrect query generation. While DA performs better by maintaining structured dialogue history, it still suffers in complex chained questions where the dependency is not linear or when entity grounding fails.

3. Logical Form Generation Errors (18%) These involve syntactically valid but semantically incorrect GQL outputs. Common examples include incorrect filtering conditions, missing relation constraints, or reversed edges. The RSE method particularly struggles here when the related schema extraction is too coarse, leading to semantically under-constrained queries.

4. Ambiguity and Underspecification (14%) These errors stem from under-specified questions, where even humans may interpret multiple valid GQLs. For example, "How about their latest investment?" may refer to different temporal orders depending on context. Models often make arbitrary choices without proper grounding, especially in ICL-AS where no external clarification mechanism exists.

5. Execution-based Errors (10%) Some errors only become evident after query execution, such as returning empty results due to overly specific filters or semantic mismatches. The DA method mitigates this partially using its GQL Refiner module, but residual issues persist due to imperfect execution feedback alignment.

Summary of Trends We observe that multi-turn interaction introduces new challenges absent in single-turn NL2GQL tasks: co-reference resolution, context propagation, and entity linking across turns are key failure points. Baselines relying on static prompts (ICL-AS) or full-schema inputs (FT-AS) tend to suffer from information overload or misalignment. Dependency-aware methods (DA) show promise but remain sensitive to entity tracking and reformulation quality.

9.9.2 Representative Error Cases

To further illustrate the limitations of baseline methods on the MTGQL dataset, we present representative error cases, highlighting how multi-turn context and schema interaction contribute to failures.

As shown in Table 12, these representative cases reveal that multi-turn NL2GQL tasks go beyond simple slot-filling. Models must integrate contextual memory, resolve references, and incorporate implicit constraints (e.g., time, status). Current baselines lack robust mechanisms for resolving such ambiguities, motivating future work toward hybrid symbolic-neural architectures or multi-agent dialogue managers.