# 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 stock in the securities industry has the highest **opening price** today?
System: **CITIC Securities.**
**match (s:stock)-[:belong_to]->(i:industry) WHERE i.name = 'securities' return s.name order by s.opening_price desc limit 1**
User: What is **its opening price** today?
System: ¥30.26
**match (s:stock {name: 'CITIC Securities'})-[:has_data]->(d:stock_data {date: '2025-01-08'}) return d.opening_price**
User: What about yesterday?
System: ¥36.25
**match (s:stock {name: 'CITIC Securities'})-[:has_data]->(d:stock_data {date: '2025-01-07'}) return d.opening_price**
User: How about **Guotai Junan Securities**?
System: ¥20.00
**match (s:stock {name: ' Guotai Junan Securities'})-[:has_data]->(d:stock_data {date: '2025-01-07'}) 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:** Leveraging our approach with a Chinese financial market NebulaGraph database, we have created the MTGQL dataset—the first Chinese multi-turn NL2GQL dataset.

- **Benchmark Methods:** We present three types of baseline methods for the MTGQL dataset, providing a solid 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.

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

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

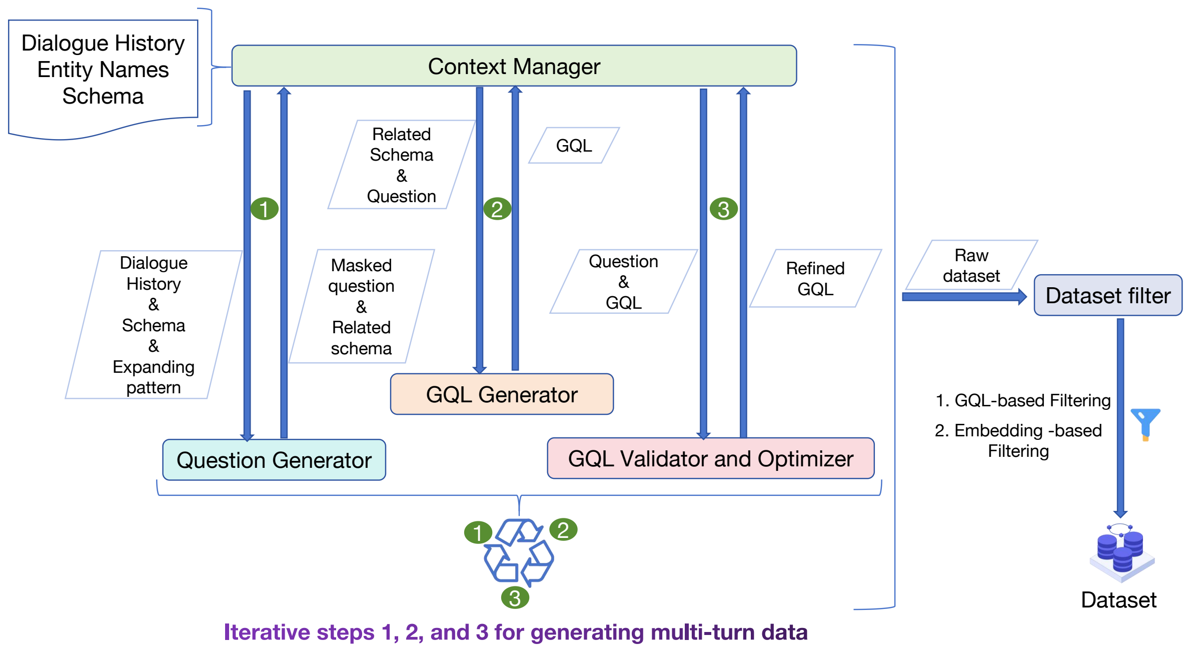
Dialogue History
Entity Names
Schema
Context Manager
Related Schema & Question
GQL
Dialogue History & Schema & Expanding pattern
Masked question & Related schema
Question & GQL
Refined GQL
Raw dataset
Dataset filter
1. GQL-based Filtering
2. Embedding -based Filtering
GQL Generator
Question Generator
GQL Validator and Optimizer
Dataset
Iterative steps 1, 2, and 3 for generating multi-turn data

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.

generate the GQL, denoted as $GQL_t$, corresponding to the question $Q_t$:

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

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 generate multi-turn NL2GQL dataset, we adhere to the following three criteria that are distinct from single-turn NL2GQL. (1) 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) 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) 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**. The Context Manager functions as the central unit, managing dialogue history, overseeing data generation, selecting appropriate dialogue dependency patterns, and filling in masked entities in questions. First, The Question Generator produces questions that are contextually coherent. Second , the GQL Generator, a fine-tuned LLM, transforms natural language into GQL. Then, the GQL Optimizer ensures correctness by performing syntax and semantic validation, correcting errors to ensure valid and accurate queries. This iterative process enables the generation of multi-turn data. 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

| 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?<br>A1: Moutai.<br>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?<br>A1: 20.5<br>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?<br>A1: HK0700<br>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?<br>A1: 150<br>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?<br>A1: 417<br>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%?<br>A1: Fund A, Fund B<br>Q2: Which ones have a size greater than 5 billion? |

Table 1: Patterns for expanding subsequent questions.

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 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 have designed the prompt format as 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. Additionally,it also provides completed versions of the colloquial questions.

## 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. It has two main functions: validating syntax and semantics, and optimizing incorrect GQL.

**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:** Numerous studies have shown that optimizing structured query statements with syntax errors can enhance their accuracy (Pourreza et al., 2024; Maamari et al., 2024; Zhou et al., 2024b). When syntax errors are detected, the system combines the original question, 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 from each entry, apply vector embedding to obtain high-dimensional representations, and calculate the similarity between data points. Any pair with similarity exceeding a preset threshold is discarded. This approach effectively reduces duplicates and enhances the uniqueness, quality, and diversity of the dataset.

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 2,the dataset includes 4500 dialogues: 3000 for training, 500 for development, and 1000 for testing. On average, each dialogue has 6.3 turns, indicating a balanced structure. It contains 21,600 GQL statements, with 14,100 in training, 2,600 in development, and 4,900 in testing. Each data point includes 4.8 GQLs, reflecting a high query density. These statistics offer valuable insights into the dataset's coverage and effectiveness for training and evaluating models in multi-turn dialogues and graph query tasks.

### 5.2 Human Evaluation

We evaluated the quality of the dataset by having three domain experts rate 200 randomly selected dialogues from the training, validation, and test sets based on coherence, question diversity, coverage, and semantic accuracy (on a 1-5 scale). The results, as shown in Table 4, demonstrate the dataset's effectiveness for training and evaluating dialogue systems.

### 5.3 Comparison with Other Datasets

As shown in Table 3, 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.

## 6 Models and Experimental Setup

### 6.1 Benchmark Methods for Multi-turn NL2GQL

We have set up three baseline methods to evaluate the performance of different strategies. The specific methods are as follows:

**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

|  | train | dev | test | total |
|---|---|---|---|---|
| Number of Data Points | 3000 | 500 | 1000 | 4500 |
| Total Number of GQLs | 18912 | 3121 | 6548 | 28581 |
| Average Dialogue Turns per Data | 6.30 | 6.25 | 6.54 | 6.35 |
| Average entity per Data | 4.72 | 5.26 | 4.97 | 4.84 |
| Average relation per Data | 5.19 | 5.33 | 5.14 | 5.19 |

Table 2: 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 | 5116 |
| 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 3: A summary of the main NL2GQL datasets. From this, we can conclude that MTGQL is the only multi-turn dataset.

|  | train | dev | test |
|---|---|---|---|
| Coherence | 4.32 | 4.18 | 4.27 |
| Question Diversity | 4.06 | 4.01 | 4.10 |
| Semantic Accuracy | 4.76 | 4.55 | 4.79 |

Table 4: Basic Statistics of The Dataset.

output, while fine-tuning the LLM. In inference, it guides the LLM to extract related schema.

**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.

## 6.2 Experimental Setup

**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 models. For sequence encoding, we utilized the all-MiniLM-L6-v2 library, with vector dimensions set to 384. All the number of demonstrations $K$ are set as 4.

**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:

$$EM = \frac{\substack{\text{number of GQL} \\ \text{with correct logic form}}}{\text{total number of GQL}} \tag{1}$$

$$\text{AEM} = \frac{\substack{\text{number of data points} \\ \text{where all GQLs have} \\ \text{correct logical form}}}{\text{total number of data points}} \tag{2}$$

$$EX = \frac{\substack{\text{number of GQL} \\ \text{with correct execution result}}}{\text{total number of GQL}} \tag{3}$$

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

| Method | Backbones | EM(%) | AEM(%) | EX(%) | AEX(%) |
|---|---|---|---|---|---|
| ICL-AS | GLM-4-9B-Chat | 32.03 | 7.80 | 30.01 | 7.10 |
| | Qwen2.5-14B-Instruct | 34.79 | 9.40 | 35.49 | 9.20 |
| | LLaMA-3.1-8B-Instruct | 29.99 | 7.20 | 30.88 | 6.70 |
| | ChatGPT-4o | 43.28 | 11.30 | 45.27 | 10.80 |
| RES | GLM-4-9B-Chat | 81.92 | 38.60 | 80.96 | 37.90 |
| | Qwen2.5-14B-Instruct | 82.59 | 40.50 | 81.98 | 39.80 |
| | LLaMA-3.1-8B-Instruct | 80.96 | 35.40 | 80.93 | 35.30 |
| FT-AS | GLM-4-9B-Chat | 84.03 | 42.30 | 83.38 | 41.60 |
| | LLaMA-3.1-8B-Instruct | 85.51 | 45.80 | 84.51 | 43.70 |
| | Qwen2.5-14B-Instruct | **86.99** | **48.80** | **85.87** | **44.30** |

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

## 7 Results

### 7.1 Main Results

Based on the results presented in Table 5, it is clear that FT-AS outperforms the other methods across all evaluation metrics, demonstrating its superior effectiveness in the given context. Specifically, FT-AS, particularly when paired with the Qwen2.5-14B-Instruct backbone, achieves the highest scores in both EM (**86.99%**) and AEM (**48.80%**), as well as EX (**85.87%**) and AEX (**44.30%**). In contrast, the ICL-AS method shows relatively lower performance, likely due to the lack of high-quality GQL-related corpora during the pretraining phase of its base models. Additionally, the comparison between different backbone models within the same method reveals noticeable performance fluctuations. This highlights the critical role of model architecture and backbone selection in influencing overall accuracy.

### 7.2 Breakdown Results by Round

| Round | EM(%) | EX(%) |
|---|---|---|
| R1 | 91.20 | 90.80 |
| R2 | 89.60 | 88.80 |
| R3 | 86.20 | 85.40 |
| R4 | 84.80 | 84.10 |
| R5+ | 80.28 | 79.96 |

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

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(%) |
|---|---|---|
| P1 | 89.26 | 89.05 |
| P2 | 84.44 | 83.98 |
| P3 | 86.52 | 85.85 |
| P4 | 91.77 | 90.93 |
| P5 | 80.47 | 78.32 |
| P6 | 82.61 | 82.29 |

Table 7: The breakdown of 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.

## 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

Although we have developed a Chinese multi-turn NL2GQL dataset, we have not yet completed the translation into English due to the need to translate much of the data in the graph database. Once this task is finished, we will release the bilingual (Chinese-English) dataset as open source.

## 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.

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.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-sql in the age of well-reasoned language models. *Preprint*, arXiv:2408.07702.

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.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

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
```

## 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 referenced 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 generate either an opening question or a follow-up question, using entity placeholders according to the expansion pattern. The output includes both a raw

**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
1  Step 1: Expansion Pattern Filtering
2  for each expansion pattern $P_i$ in $\{P1, P2, \ldots, P6\}$ do
3      if Pattern $P_i$ meets the predefined conditions based on E, R, and G then
4          Include $P_i$ in the set of valid patterns
5      else
6          Remove $P_i$ from the set of valid patterns
7  Step 2: Expansion Pattern Selection
8  for each valid expansion pattern $P_i$ do
9      Set initial weight of $P_i$ as $w(P_i) = \frac{1}{6}$
10 for each previously used expansion pattern $P_i$ do
11     Halve its weight: $w(P_i) = \frac{w(P_i)}{2}$
12     Redistribute the halved weight equally among other remaining patterns
13 Select the expansion pattern $P_{\text{selected}}$ with the highest weight:
14 $P_{\text{selected}} = \arg\max_{P_i} w(P_i)$
15 Step 3: Entity and Relation Selection
16 Determine the potential candidate entities $E_{\text{candidates}}$ based on $P_{\text{selected}}$
17 for each candidate entity $e \in E_{candidates}$ do
18     Set initial weight of entity $e$ as $w(e) = \frac{1}{|E_{\text{candidates}}|}$
19     if $e$ has been referenced in the previous dialogue step then
20         Increase $w(e)$ by $\frac{1}{4}$, indicating higher likelihood of selection
21     Redistribute the increased weight evenly among other remaining entities
22 Determine the potential relations $R_{\text{candidates}}$ based on $P_{\text{selected}}$
23 for each relation $r \in R_{candidates}$ do
24     Assign weight to $r$ using a similar process as entity selection
25 return Selected expansion pattern $P_{selected}$, selected entities, and selected relations
```

question with references and a fully disambiguated version, free of placeholders and references, ensuring both contextual relevance and structural clarity. It is worth noting that since we are constructing a Chinese dataset, the prompt is written in Chinese. However, for ease of reading, we have translated it into English.

**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 the question is contextually relevant and continues the conversation logically.
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.