A Comprehensive LLM-based Pipeline and Benchmark for Text-to-ES Task

Anonymous ACL submission

Abstract

Elasticsearch (ES) is a distributed RESTful search engine that is well-suited for managing large-scale data and long-text search, outperforming relational databases in these areas. Recent research on Text-to-Query has explored using large language models (LLMs) to convert user query intent into executable code, making it an increasingly popular research topic. However, the ES query language, domain-specific language (DSL), poses challenges due to its complex JSON structure, making it difficult for users to manually create queries. Despite the growing interest in this area, there remains a lack of effective research addressing these challenges. Furthermore, unlike SQL, ES cannot rely solely on its DSL to fulfill user needs in cross-index scenarios due to its NoSQL limitations.To address these challenges, in this paper, we present a comprehensive Text-to-ES pipeline that transforms natural language into DSL queries. Moreover, we propose the ES Cross-index Query paradigm to handle Cross-index Query scenarios effectively. In addition, we introduce two datasets, LED and BirdES, to benchmark LLM performance on the Text-to-ES task. The LED dataset encompasses nearly all official query types, totaling 224.9 GB with 26,207 Text-ES pairs, whereas the BirdES dataset, derived from the Bird dataset, features 10,962 Text-ES pairs focused on Cross-index Query scenarios,which is closer to the real-world. Finally, we trained a Text-to-ES model, evaluated the performance of advanced large models on LED and BirdES, and provided a detailed analysis of the results. We will release our code and dataset in the future.

1 Introduction

Elasticsearch (ES) is a distributed RESTful search engine (Akdal et al., 2018b) that offers powerful full-text search capabilities, allowing it to process petabytes of data in seconds[1]. It effectively addresses challenges related to scalability, large-scale data search, and performance issues that relational databases are inherently unable to handle (Kononenko et al., 2014).

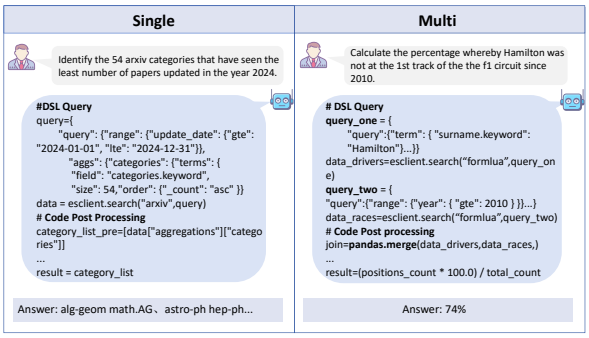

Figure 1: Example of transforming natural language into Elasticsearch query statement, the left graph shows the case when the user question involves only one index corresponding to the LED dataset. The right figure shows the case when the user question involves multiple indexes corresponding to the BirdES dataset.

Text-to-Query refers to using LLM to convert user intent into executable code. Currently, the most rapidly developing area is Text-to-SQL (Hong et al., 2024b), which transforms natural language into SQL queries. Representative works in this field include WikiSQL (Zhong et al., 2017), Spider (Yu et al., 2018), and Bird (Li et al., 2024b). Similarly, Text-to-Cypher (Guo et al., 2022) focuses on the automated generation of knowledge graph Cypher query, alongside related processes such as Text-to-CQL (Lu et al., 2024) and Text-to-SPARQL (Yin et al., 2021). However, the ES query language, known as Domain Specific Language (DSL), presents challenges due to its complex JSON structure, making it difficult for users to manually construct queries. Previous research (Akdal et al., 2018b) has utilized model-driven development (MDD) (Selic, 2003; Porubän et al., 2014) approaches to generate ES DSL queries, but this approach is limited to specific rules and lacks

[1] https://www.elastic.co/cn/elasticsearch

flexibility. Furthermore, ES has the NoSQL limitation of not being able to perform cross-index queries. Unlike SQL, ES cannot use its DSL for cross-index queries; it requires post-processing to facilitate these operations.

This paper introduces the task of converting natural language into ES query statements using LLM, referred to as Text-to-ES, as illustrated in Figure 1. To address the challenges of writing ES queries, we collected data from Wikipedia and Kaggle to create LED, a large-scale ES database grounded in Text-to-ES, containing 26,207 Text-ES pairs with a total size of 224.9 GB. Furthermore, to tackle the issue of ES's lack of support for Cross-index Query, we propose the ES Cross-index Query paradigm and constructed the BirdES dataset, based on Bird, which features numerous cross-table queries. BirdES comprises 10,962 Text-ES pairs, with nearly 80% Cross-index Query data and a highly complex mapping structure. Finally, we trained a large language model (LLM) on LED and BirdES and evaluated state-of-the-art LLMs on both datasets, providing a detailed analysis of the experiments.Our contribution is as fllowing:

- **Proposing a Comprehensive Text-to-ES Pipeline Based on LLMs:** We proposed a Text-to-ES task based on LLMs. Additionally, to enable ES to support cross-index query, we introduced the Cross-index Query Paradigm, which allows ES to implement cross-index query. These two parts together form a comprehensive Text-to-ES pipeline, as shown in Figure 1.

- **Proposing the LED and BirdES Datasets for Text-to-ES:** Figure 2 illustrates the detailed construction process of LED and BirdES. The LED dataset encompasses nearly all query types released in the official ES documentation. BirdES is a more complex dataset that closely resembles real-world application scenarios and contains a substantial number of cross-index query.

- **Conducting Comprehensive Evaluation and Analysis Experiments:** We trained a Text-to-ES model and conducted extensive evaluation and analysis experiments on the LED and BirdES datasets using advanced LLMs. We further categorized the data to provide a comprehensive analysis of the result across different categories.

2 Text-to-ES Pipline

Text-to-ES refers to the process of converting a natural language question Q into a DSL query D and a post-processing code C. That is capable of retrieving data corresponding to the query question Q from ES. Each ES index has its own mapping structure M. The M can be represented as follows:

$$M = \{F_i : (T_i, P_i)\}_{i=1}^{n}$$

In this context, F_i denotes the name of the $i-th$ field, T_i represents the data type of the $i-th$ field, and P_i refers to the configuration properties of the $i-th$ field (optional, such as keyword, storage, etc.). This mapping structure significantly aids the model in understanding the ES database, leading to more accurate responses. The task definition of Text-to-ES is thus formulated as follows:

$$\mathbf{R} = f(\mathbf{Q}, \mathbf{M} \mid \boldsymbol{\theta})$$

The function $f(\cdot \mid \boldsymbol{\theta})$ represents a model with $\boldsymbol{\theta}$ parameters, where R denotes the model's response based on Q and M. This response, R, consists of two components: D and C. By executing R, one can retrieve data that meets the requirements of Q.

3 Releated Work

3.1 Text-to-Query

Text-to-Query is the process of using LLM to convert user intent into executable executable code. Firstly, Text-to-SQL based on large language models (LLMs) is mainly divided into two categories. The first category is GPT-based frameworks for Text-to-SQL. Notable examples are DEA-SQL (Xie et al., 2024), which employs a complex pipeline to enhance accuracy, alongside DIN-SQL (Pourreza and Rafiei, 2024a), MBR-Exec (Shi et al., 2022), Coder-Reviewer (Zhang et al., 2023b), LEVER (Ni et al., 2023), SELF-DEBUGGING (Chen et al., 2023), StructGPT (Jiang et al., 2023), Least-to-Most (Tai et al., 2023), and DELLM (Hong et al., 2024a).The second category is that enhances the text-to-SQL process through training models. Representative works include CodeS (Li et al., 2024a), which compiles extensive SQL-related data during its pre-training phase. Other similar works include Granite (Mishra et al., 2024) , CLLM (Kou et al., 2024), DAIL-SQL (Gao et al., 2023), Symbol-LLM (Wu et al., 2024), StructLM (Zhuang et al., 2024), and DTS-SQL (Pourreza and Rafiei, 2024b).In the field

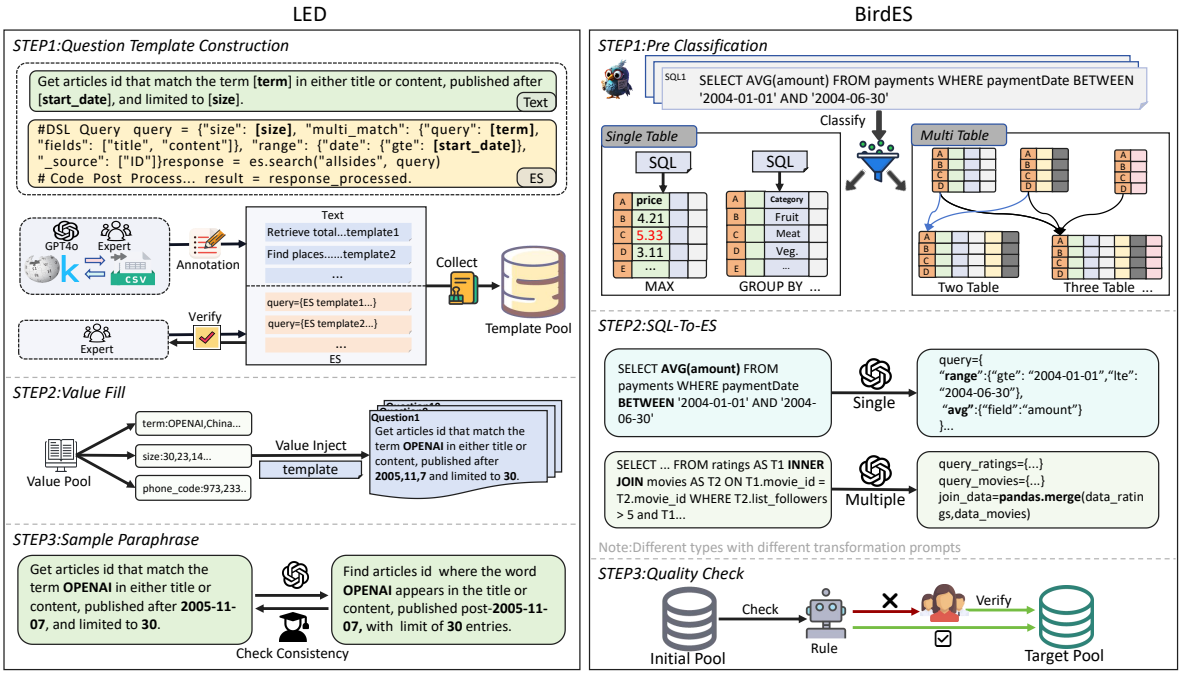

Figure 2: Detailed flowchart of data construction. On the left, the LED data construction process is depicted, where Text-ES template pairs are created using multiple experts in collaboration with GPT. The templates are populated with values from the database, and the constructed data is rewritten using GPT. On the right, the BirdES data construction process is illustrated, where SQL statements from the Bird dataset in the Text-to-SQL domain are transformed into ES queries to build BirdES, which is subsequently evaluated.

of Text-to-Cypher, the first dataset, SpCQL, was proposed by (Guo et al., 2022). Additional contributions in this area include works such as (Zhao et al., 2022, 2023a; Liang et al., 2024; Zhao et al., 2023b). Beyond these two domains, notable efforts include Text-to-CQL (Lu et al., 2024), which transforms natural language into corpus query statements, and Text-to-SPARQL (Soru et al., 2017; Luz, 2019; Jung and Kim, 2020; Yin et al., 2021), which converts natural language into SPARQL query statements. We are the first to propose the Text-to-ES task in the Text-to-Query domain, which bridges the gap in this domain in terms of automatic querying of ES database.

3.2 Domain Structured Language Generation

DSL is a programming or scripting language designed for specific application domains. Before LLMs emerged, (Akdal et al., 2018a) used Model-driven techniques to generate Query DSL for ES query, differing from modern machine learning models by focusing on code that maps to a DSL. At present, LLMs excel in generating code for languages like C, C++, and Python but struggle with DSLs. For instance, (Bassamzadeh and Methani, 2024) utilized retrieval augmentation for Web API DSLs, while autoDSL (Shi et al., 2024) created a framework for generating DSLs for non-displayed queries with LLMs, especially for non-standard experimental constraints. Although (Akdal et al., 2018a) shares a similar motivation, it has limitations due to its reliance on user interaction and specific requirements. In contrast, we propose an advanced LLM-based Text-to-ES pipeline, which serves as a more standardized approach for the automated generation of ES query.

4 DataConstruction

4.1 LED

The LED dataset encompasses nearly all commonly used DSL from the official Elasticsearch documentation.

4.1.1 Template Construct

The template construction aims to create numerous Text-ES template pairs, as shown in Figure 2, based on index mapping information and the DSL types from the official Elasticsearch documentation. We collected a substantial amount of long-text data from open data platforms such as Kaggle and Wikipedia. Additionally, we incorporated NASA's geographic data to leverage Elasticsearch's geo-

Dataset	# Examples	# Index	# Row/Index	# Column/Index	# Size	Domain	Schema-Free
WikiSQL	80,664	24,241	0.01k	6	0.2GB	SQL	×
Spider	96,913	1,020	1k	5	1.7GB	SQL	×
Bird	12,751	693	530k	4	33.4GB	SQL	×
LED	26,207	34	88k	37	224.9GB	ES	✔
BirdES	10,192	693	530k	4	33.4GB	ES	×

Table 1: The Compresion between LED,BirdES and other Text-to-SQL DataSet.

graphic query capabilities. To our knowledge, existing Text-to-SQL datasets do not include features targeting geographic queries. After data collection was completed, multiple Elasticsearch experts and GPT collaboratively created Text-ES template pairs based on all the index mapping information. Once the creation process was finished, the Elasticsearch experts conducted deduplication and error correction to refine the templates, ultimately yielding 2,600 Text-ES template pairs.

4.1.2 Value Filling

The value filling step is designed to insert appropriate values into the created templates to form Text-ES pairs. We employed both automated data retrieval and manual input methods. For non-open fields, such as names and geographical locations, we directly extracted relevant data from the index. For open fields, such as title keywords, values were manually crafted based on the index data. Through these two approaches, we developed a Value Pool. By inputting a Text-ES template into the Value Pool, we generated approximately ten Text-ES pairs.

4.1.3 Question Rewrite

The question rewrite step is intended to enhance the semantic richness of the LED data. Some semantic redundancy occurs in the data generated by template construction in the previous phase. To address this, we carefully rewrite a portion of the problems as In- Context Learning(ICL) (Dong et al., 2022) examples, providing clearer guidance for subsequent rewrites in GPT-4o (OpenAI, 2024), ultimately improving the overall quality and diversity of the generated outputs.

4.1.4 Quality Check

In the quality assessment phase, we concentrated on two key dimensions: consistency and readability of the rewritten questions. In terms of consistency, we rigorously evaluate whether the rewritten questions align with the corresponding ES query statements. In terms of readability, our focus is on whether the logical structure of the rewritten questions is clear and coherent. We employed a random sampling method, extracting 1,000 samples from the dataset in three rounds for review. If over 90% of the samples meet both completeness and readability standards, they are deemed approved.

4.2 BirdES

The BirdES dataset is derived from the Text-to-SQL dataset Bird. It was proposed to create a Text-to-ES dataset that more closely reflects real-world scenarios and includes cross-index queries.

4.2.1 Pre-Classification

The pre-classification step is designed to categorize the SQL data into different classes. Use different transformation strategies based on the specific categories. If a SQL contain only one table, it is classified as a single-table query. Conversely, it is classified as a multi-table query. Moreover, Single-table queries are further categorized into fifteen distinct groups based on specific keywords in the query, such as max, like, group by and etc. Similarly, multi-table queries are further subdivided into eight distinct groups based on the number of tables involved, with queries involving two tables being the most numerous.

4.2.2 Single Table Conversion

The purpose of the single table conversion step is to transform single-table SQL query data into the corresponding ES DSL query. SQL query statements follow a fixed execution order. In transformation process, we leverage prompt engineering to map MySQL queries to corresponding ES queries. The WHERE clause corresponds to the Pre-Process stage, which is analogous to the "query" section in a DSL query, responsible for filtering documents. The GROUP BY and HAVING clauses represent the Intermediate-Process stage, equivalent to the "aggs" (aggregations) part in a DSL query, which handles data aggregation. The SELECT clause

corresponds to the Post-Process stage, akin to the "_source" section of a DSL query, determining the final output fields. Special functions in MySQL, such as "CAST" and "CASE," require handling through post-processing code.

4.2.3 Multiple Table Conversion

The multiple table conversion step is designed to transform multi-table SQL query data into cross-index Elasticsearch query code. ES does not inherently support cross-index queries due to its NoSQL limitations.Initially, we attempted to address this limitation by converting multi-table query into multiple single-table queries, exploring various methods to solve the problem. We later realized that this approach was not feasible, and (Zhang et al., 2023a) was constrained by this limitation as well.Ultimately, we use the merge method from the Pandas library in the post-processing code to dynamically merge the Index data involved in cross-index queries based on the specific question.

4.2.4 Quality Check

The quality check step focuses on primarily verifying whether the execution results of the original SQL and Elasticsearch are consistent. There are situations when the results of the two are inconsistent, yet the ES query remains correct; this is due to differences in their design. This situation requires further confirmation by the annotator. As long as the DSL align with the question intent, we judge the ES query as correct. Ultimately, we performed three random samplings of 1,000 entries each, achieving an accuracy rate exceeding 80%.

4.3 Data Statistics

Table 1 presents an overview of Text-to-ES datasets, including LED and BirdES, in comparison with well-known Text-to-SQL domain datasets.The total size of the LED index data is 224.9 GB, containing 26,207 Text-ES pairs. LED includes 23,099 train samples, 1,569 dev samples, and 1,539 test samples. LED has a higher average number of fields per index, with approximately 37 fields per index. Furthermore, unlike SQL datasets with fixed schema, many real-world scenarios involve schema-free situation. LED is the first dataset that incorporates data with such schema-free formats.The total size of the BirdES index data is 33.4 GB, containing 10,192 Text-ES pairs. Nearly 80% of the data consists of cross-index queries. The BirdES dataset includes 9,428 train samples and 1,534 test samples.

5 Experiment

Table 1 presents our construction of the Text-to-ES domain datasets, LED and BirdES, in comparison with well-known Text-to-SQL domain datasets.LED has a higher average number of fields per index, with approximately 37 fields per index. The total data size reaches 224.9 GB, characterized by large-scale long text. Furthermore, unlike SQL datasets with fixed schema, many real-world scenarios involve schema-free situation. LED is the first dataset that incorporates data with such schema-free formats.

5.1 Baselines

We primarily utilized both advanced models and code models.

5.1.1 Advanced Model

In the general series of models, we primarily focused on the following models, including LLaMA2-7B, LLaMA2-13B (Touvron et al., 2023), and their Chat versionsïijŇas long as LLaMA3.1-8B-Instrcut and LLaMA3-8B-Instrcut (Dubey et al., 2024). Furthermore, we expanded our experiments to incorporate other advanced open-source models, such as Qwen2.5-7B-Instruct,Qwen2.5-14B-Instruct (Yang et al., 2024). In addition, we also used GPT-4o (OpenAI, 2024) as our evaluation model.

5.1.2 Code Model

To evaluate the performance of code models on Text-to-ES tasks, we selected various code-focused models corresponding to the previously mentioned open-source counterparts. These include models such as CodeLLaMA-7B,CodeLLaMA-13B (Roziere et al., 2023),Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), and deepseek-coder-6.7-base (Guo et al., 2024).

5.2 Implementation Details

We deployed different models on 16 NVIDIA A100 40GB GPUs. All experimental results are based on a few-shot setup, as the models exhibited certain limitations in following instructions and formats under the zero-shot configuration A1. We set the temperature to 0.01 to minimize randomness as much as possible.

5.3 Evaluation

Domain Structured Language Exact Match Accuracy (DSLEM) refers to the measure of whether the DSL in a generated query precisely matches the DSL query in the ground truth. The calculation formula is as follows:

$$\text{DSLEM} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(Q_n, \hat{Q}_n)$$

The ground truth DSL be represented as Q_n and the generated DSL as $\hat{Q}_n$. If Q_n exactly matches $\hat{Q}_n$, The function $\mathbb{1}$(Âů) is a decision function used to determine whether Q_n and $\hat{Q}_n$ are equal. The detailed calculation process is provided in Appendix B.1.

Execution Accuracy (EX) EX refers to the exact match between the model-generated query code and the ground thrth result. This metric is more stringent than DSLEX, as even if the DSL is correctly generated, generating the correct post-processing code remains a challenging task.The formula is shown below:

$$\text{EX} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(O_n, \hat{O}_n)$$

The terms O_n and $\hat{O}_n$ represent the final output of the ground truth query code and the model-generated query code, respectively. The function $\mathbb{1}$(Âů) is used to determine whether O_n and $\hat{O}_n$ are identical, with the detailed calculation process provided in Appendix B.2.

Valid Efficiency Score (VES) VES is designed to evaluate the execution efficiency of the generated ES query. The specific calculation process is outlined below:

$$\text{VES} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(V_n, \hat{V}_n) \cdot \mathbf{R}(Y_n, \hat{Y}_n)$$

Here, $\hat{Y}_n$ and $\hat{V}_n$ represent the ES query generated by the model and its corresponding execution result, while Y_n and V_n denote the ground truth ES query and its execution result. The function $\mathbb{1}$(Âů) is used to determine whether V_n and $\hat{V}_n$ are equal. R(Âů) is a function that evaluates the efficiency ratio. Further details can be found in the derivation of formulas section of Appendix B.3.

6 Analysis

6.1 Main Experiment

Table 2 presents the performance of different models on the LED and BirdES datasets. Among them, GPT4o achieves the best results overall. The experimental results on the LED dataset indicate that LLaMA2-13B-Chat ranks just below GPT4o within the Advanced models. Notably, Qwen2.5-14B-Instruct only achieves an EX score of 8.51%. In the Code models, CodeLLaMA-13B-Instruct demonstrates strong performance, with an EX score of 26.19%. Additionally, fine-tuning CodeLLaMA-13B-Instruct on the LED dataset results in a 6.62% increase in EX and a 6.96% increase in VES. The experimental results on the BirdES dataset reveal that the same models perform significantly worse than on the LED dataset, with GPT4o achieving only 25.75% in EX. Similarly, models fine-tuned on BirdES see only a 1.37% improvement in DSL-EM.

Base VS Chat. The comparison of the four modelsâĂŤLLaMA2-7B, LLaMA2-7B-Chat, LLaMA2-13B, and LLaMA2-13B-ChatâĂŤreveals several key insights. Within the same base class or Chat class, models with a larger parameter size demonstrate superior performance. Specifically, LLaMA2-13B achieves a significantly higher EX metric score of 23.07%, compared to LLaMA2-7B's 4.22%. Furthermore, for models with the same parameter size, the Chat models consistently outperform their corresponding Base models, which have not undergone fine-tuning. Notably, LLaMA2-13B-Chat surpasses LLaMA2-13B across all three evaluation metrics.

Chat VS Code. By comparing the results of LLaMA2-7B-Chat and LLaMA2-13B-Chat with CodeLLaMA-7B-Instruct and CodeLLaMA-13B-Instruct, it is evident that CodeLLaMA, which is derived from further fine-tuning of LLaMA2, exhibits improved performance. As shown in Table 2, the fine-tuned CodeLLaMA-13B-Instruct consistently outperforms LLaMA2-13B-Chat across all three evaluated metrics.

6.2 Different types of ES queries

We categorized the LED dataset into six types: Specialized, TermLevel,FullText, Geography, Joining,and Aggregation.

Table 3 shows that GPT-4o achieves state-of-the-art (SOTA) performance in all categories. In addition, The Feynman-13B achieves sub-optimal

Models	LED			BirdES		
	DSL-EM	EX	VES	DSL-EM	EX	VES
Advanced Models						
LLaMA2-7B (Touvron et al., 2023)	0.78	4.22	4.26	0	0	0
LLaMA2-7B-Chat (Touvron et al., 2023)	4.48	5.07	5.06	0	0	0
LLaMA3-8B-Instruct (Dubey et al., 2024)	3.37	6.04	5.85	0	0.07	0.27
LLaMA3.1-8B-Instruct (Dubey et al., 2024)	4.74	7.08	7.09	0	0	0
Qwen2.5-7B-Instruct (Yang et al., 2024)	4.48	7.67	7.62	0	0.71	0.96
Qwen2.5-14B-Instruct (Yang et al., 2024)	0.90	8.51	9.10	0	0.26	0.33
LLaMA2-13B (Touvron et al., 2023)	12.02	23.07	23.30	0	0.39	0.72
LLaMA2-13B-Chat (Touvron et al., 2023)	13.84	24.37	24.51	0	0	0
GPT-4o (OpenAI, 2024)	**30.15**	**48.73**	**49.42**	**2.35**	**25.75**	**35.12**
Code Models						
Qwen2.5-Coder-7B-Instruct (Hui et al., 2024)	0.25	5.01	5.01	0.07	0.39	1.04
CodeLLaMA-7B-Instruct (Roziere et al., 2023)	0.37	5.79	6.20	0.13	0.52	0.43
Deepseek-coder-6.7b-base (Guo et al., 2024)	5.52	8.97	9.08	0	0	0
CodeLLaMA-13B-Instruct (Roziere et al., 2023)	19.42	26.19	26.20	0	0.39	0.72
Ours						
FeynMan-13B	20.92	32.81	33.16	1.37	0.26	0.27

Table 2: Performance comparison on LED and BirdES benchmarks

Models	Specialized		TermLevel		FullText		Geography		Joining		Aggregation	
	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX
GPT-4o (zero-shot)	0	33.33	3.70	28.34	10.04	35.20	1.40	14.05	3.15	15.98	1.73	34.63
GPT-4o (few-shot)	**28.00**	**48.58**	**34.09**	**50.92**	**20.80**	**38.40**	**51.87**	**62.15**	**33.41**	**45.04**	**18.61**	**46.32**
Qwen2.5-7B-Instruct	1.45	0.73	8.01	13.76	14.40	25.60	0.47	3.27	1.45	0.73	1.73	2.16
Qwen2.5-14B-Instruct	0.00	0.00	1.85	15.81	0.80	8.00	0.00	5.14	0.97	2.91	0.00	9.09
CodeLLama-13B-instruct	10.14	15.94	13.76	31.01	6.40	16.70	20.09	23.36	7.26	14.04	13.85	26.41
FeynMan-13B	26.09	33.33	27.31	39.84	7.20	16.00	17.29	22.43	21.31	31.72	16.02	38.53

Table 3: LED different categories of models on each model performance effect.

results in almost all categories. Feynman-13B, after fine-tuning, shows improvements across all six categories; however, there remains a noticeable gap in performance compared to GPT-4o. It is worth noting that there is not much difference in performance between Qwen-2.5-7B-Instruct and Qwen-2.5-14B-Instruct.Additionally, GPT-4o achieved an almost 50% improvement in Geography after adopting a few-shot setup, and the reasons behind this enhancement merit further investigation.

6.3 BirdES Single vs. BirdES Multiple Tables

Models	Single			Multiple		
	DSLEM	EX	VES	DSLEM	EX	VES
GPT-4o(zero-shot)	2.8	11.4	12.4	0.4	8.5	12.1
GPT-4o(few-shot)	2.8	34.5	36.5	2.2	13.6	21.9

Table 4: Experimental results of BirdES single and multi-table type data

We analyzed the model's performance in comparing single-index and cross-index queries. The queries were classified based on the number of indexes involved: queries with only one index were labeled as single-index queries, while those involving multiple indexes were categorized as cross-index queries. Using the few-shot approach, GPT-4o demonstrated a 20.1% improvement in EX metrics and a 24.1% improvement in VES metrics for the single-index category. Furthermore, in the multi-index category, it improved EX metrics by 5.1% and VEW metrics by 9.8%.

6.4 Comparison effect of different difficulty levels of BirdES

BirdES directly adopts the difficulty classification used in Bird. This section explores the correctness and execution rates of the model-generated results, thus focusing solely on the EX and VES metrics. As illustrated in Figure 3, few-shot learning enhances the performance across all three difficulty levels. First, the results for few-shot learning sur-

	LED
Question	Find articles that mention "to" in the title and were published in 2014 under the topic "us_house"?
Golden	query={"title":"Gowdy","script":"2014","topic":"us_house","_source":"ID"}
GPT4o	query={"title":"Gowdy","script":"2014","topic":"us_house","_source":"ID"}
Qwen2.5-14B-Instruct	query={"title":"*Gowdy*","range":"2014-2015.01.01","topic":"us_house","aggs":"id",_source":"ID"}
FeynMan-13B	query={"ID":"Gowdy","script":"2014","topic":"us_house","_source":"ID,date,topic"}
	BirdES
Question	Chlorine is in what type of bond?
Golden	query1={"element":"cl"}, query2={"atom_id":atom_id_list}, query3={"bond_id":bond_id_list}
GPT4o	query1={"element":"Chlorine"}, query2={"atom_id":atom_id_list}, query3={"bond_id":bond_id_list}
Qwen2.5-14B-Instruct	query1={"element":"Chlorine"}, query2={"atom_id":atom_id_list}, query3={"bond_id":bond_id_list}
FeynMan-13B	query1={"label":"Cl"}, query2={"molecule_id":molecule_id}, query3=null

Table 5: Error analysis for different models on LED and BirdES

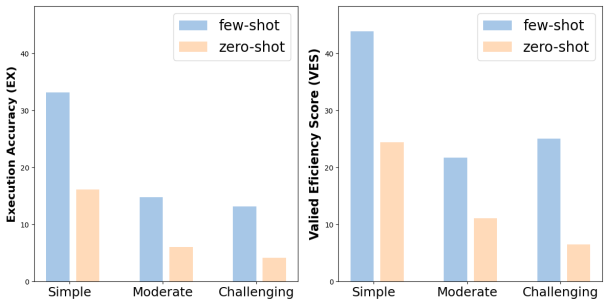

Figure 3: Results at Simple, Moderate, and Challenging difficulties for BirdES

pass those of zero-shot learning, which shows that the model's understanding of the Text-to-ES task is insufficient without sample data. The left graph reveals that accuracy declines as the difficulty increases, while the right graph shows that the efficiency taken by the model-generated queries does not exhibit a similar trend. Simple queries take relatively little time, leaving limited room for improvement. In contrast, there is significant potential for enhancing more complex queries, which is why the VES metric actually improves under the Challenging category.

6.5 Error Analysis

GPT4o achieves the best results on both datasets, with errors primarily arising from the inability to establish the correct relationship between the question and the associated index mapping structure. Additionally, GPT4o's poor performance on cross-index queries may be partly due to the requirement for external knowledge to accurately answer certain queries. For example, in the case presented in Table 5, the question mentions "Chlorine," but the correct query should be "cl". This result relies heavily on external knowledge. Another possible reason is that GPT4o struggles to construct the correct query logic based on the question.**Qwen2.5-14B-Instruct** performs poorly on both datasets, likely because Text-to-ES is a new task, making it challenging for existing models to generate ES code. Furthermore, the index mapping structure in BirdES is highly complex, resulting in lengthy contexts during model inference.Although **FeynMan-13B** shows improvement on both datasets after training, its performance still does not surpass that of GPT4o. This may be attributed to the limitations of the base model's capabilities and parameter size.

7 Conclusion

In this paper, we present a comprehensive Text-to-ES pipeline that transforms natural language into DSL queries. Moreover, we propose the ES Cross-index Query paradigm to handle cross-index scenarios effectively. Additionally, we constructed two Text-to-ES datasets: LED, which covers nearly all commonly used query types from the official Elasticsearch documentation, and BirdES, which is closer to real-world scenarios and includes substantial cross-index queries. We bridge a gap in the research on LLMs in the Text-to-ES domain. Results from various models on the LED and BirdES datasets highlight significant challenges in the Text-to-ES task, indicating ample room for improvement and innovation.

Limitation

1)The LED dataset still has gaps compared to real ES applications, and the fine-tuned FeynMan-13B model has not yet achieved satisfactory performance. 2)Although BirdES closely resembles real-world scenarios, the improvement in the fine-tuned model over the non-fine-tuned model is minimal, and the reasons for this slight enhancement warrant further exploration. 3)Our experimental design is not comprehensive enough; we should explore the results of fine-tuning various types of models on our dataset. 4)We should further explore the model performance under different categorizations of the LED task. 5)Our evaluation method for the task has limitations; since Text-to-ES involves multiple intermediate processes, a more comprehensive assessment should be provided. 6)The dataset contains a certain amount of noisy data, and further exploration of its impact on the task is required.

Ethical Considerations

Our dataset does not involve any task privacy issues. Additionally, the dataset was verified by ES experts to ensure high quality. We will be releasing the dataset publicly for research purposes in the future. To the best of our knowledge, we are not aware of any other potential ethical implications of the proposed dataset.

References

Berkay Akdal, Zehra Gül Çabuk Keskin, Erdem Eser Ekinci, and Geylani Kardast. 2018a. Model-driven query generation for elasticsearch. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 853–862. IEEE.

Berkay Akdal, Zehra GÃijl ÃĞabuk Keskin, Erdem Eser Ekinci, and Geylani Kardast. 2018b. Model-driven query generation for elasticsearch. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 853–862.

Nastaran Bassamzadeh and Chhaya Methani. 2024. Plan with code: Comparing approaches for robust nl to dsl generation. *arXiv preprint arXiv:2408.08335*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

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*.

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

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*, pages 3973–3977.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024a. Knowledge-to-sql: Enhancing sql generation with data expert llm. *arXiv preprint arXiv:2402.11517*.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024b. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Haemin Jung and Wooju Kim. 2020. Automated conversion from natural language query to sparql query. *Journal of Intelligent Information Systems*, 55(3):501–520.

Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2014. Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 328âĂŞ331, New York, NY, USA. Association for Computing Machinery.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. 2024. Cllms: Consistency large language models. *arXiv preprint arXiv:2403.00835*.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards

building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Yuan-Lin Liang, Chih-Yung Chang, and Shih-Jung Wu. 2024. Kei-cql: A keyword extraction and infilling framework for text to cypher query language translation. *International Journal of Design, Analysis & Tools for Integrated Circuits & Systems*, 13(1).

Luming Lu, Jiyuan An, Yujie Wang, Cunliang Kong, Zhenghao Liu, Shuo Wang, Haozhe Lin, Mingwei Fang, Yaping Huang, Erhong Yang, et al. 2024. From text to cql: Bridging natural language and corpus search engine. *arXiv preprint arXiv:2402.13740*.

Fabiano Ferreira Luz. 2019. *Deep neural semantic parsing: translating from natural language into SPARQL*. Ph.D. thesis, Universidade de São Paulo.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128. PMLR.

OpenAI. 2024. Hello gpt-4o. Technical report, OpenAI.

Jaroslav Porubän, Michaela Bačíková, Sergej Chodarev, et al. 2014. Pragmatic model-driven software development from the viewpoint of a programmer: Teaching experience. In *2014 Federated Conference on Computer Science and Information Systems*, pages 1647–1656. IEEE.

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

Mohammadreza Pourreza and Davood Rafiei. 2024b. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Bran Selic. 2003. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I Wang. 2022. Natural language to code translation with execution. *arXiv preprint arXiv:2204.11454*.

Yu-Zhe Shi, Haofei Hou, Zhangqian Bi, Fanxu Meng, Xiang Wei, Lecheng Ruan, and Qining Wang. 2024. Autodsl: Automated domain-specific language design for structural representation of procedures with constraints. *arXiv preprint arXiv:2406.12324*.

Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. 2017. Sparql as a foreign language. In *SEMANTiCS (Posters & Demos)*.

Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain-of-thought style prompting for text-to-sql. *arXiv preprint arXiv:2305.14215*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xiaoqian Wu, Yong-Lu Li, Jianhua Sun, and Cewu Lu. 2024. Symbol-llm: leverage language models for symbolic system in visual human activity reasoning. *Advances in Neural Information Processing Systems*, 36.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, MingXiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, ChengXiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm. *arXiv preprint arXiv:2402.10671*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. 2021. Neural machine translating from natural language to sparql. *Future Generation Computer Systems*, 117:510–519.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023a. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. *arXiv preprint arXiv:2310.17342*.

Tianyi Zhang, Tao Yu, Tatsunori Hashimoto, Mike Lewis, Wen-tau Yih, Daniel Fried, and Sida Wang. 2023b. Coder reviewer reranking for code generation.

In *International Conference on Machine Learning*, pages 41832–41846. PMLR.

Ziyu Zhao, Wei Liu, Tim French, and Michael Stewart. 2023a. 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, Wei Liu, Tim French, and Michael Stewart. 2023b. Rel2graph: Automated mapping from relational databases to a unified property knowledge graph. *arXiv preprint arXiv:2310.01080*.

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

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

Alex Zhuang, Ge Zhang, Tianyu Zheng, Xinrun Du, Junjie Wang, Weiming Ren, Stephen W Huang, Jie Fu, Xiang Yue, and Wenhu Chen. 2024. Structlm: Towards building generalist models for structured knowledge grounding. *arXiv preprint arXiv:2402.16671*.

A Details of Our Datasets

A.1 Text-to-ES

Text: Identify the names and IDs of meteorites that belong to the class H4 and have a longitude between 138.187 and -88.049.

ES:
```
# Query DSL
query = {
  "query": {
    "bool": {
      "must": [
        {"term": {"class.keyword": "H4"}},
        {"geo_bounding_box": {
            "geolocation": {
              "top_left": {
                "lat": 90.0,
                "lon": 138.187},
              "bottom_right": {
                "lat": -90.0,
                "lon": -88.049}}
          }
        }
      ]
    }},
  "_source": ["name", "id"]
}
index_name = "nasa_meteorites"
# Post Process Code
response = esclient.search(index=index_name, body=query)
result = [(hit['_source']['name'], hit['_source']['id']) for hit in response['hits']['hits']]
```

Figure A1: Example of LED Text-to-ES

Text: Provide a list of titles together with its publisher name for all publishers located in the USA.

ES:
```
# Query DSL1
# Step 1: Query the `book_publishing_company_publishers` index to get documents where `country` is 'USA'
query_publishers = {
  "query": {"term": {"country.keyword": "USA"}}}
response_hits = [doc["_source"] for doc in scan(client=esclient, index="book_publishing_company_publishers", body=query_publishers,
size=10000)]
# Extract fields for future use
publishers_data = [(hit["pub_id"], hit["pub_name"]) for hit in response_hits]

# Convert to DataFrame for ease of handling
publishers_df = pd.DataFrame(publishers_data, columns=["pub_id", "pub_name"])
# Query DSL2
# Step 2: Query the `book_publishing_company_titles` index to get all documents
query_titles = {
  "query": {"match_all": {}}}
response_hits = [doc["_source"] for doc in scan(client=esclient, index="book_publishing_company_titles", body=query_titles, size=10000)]
# Extract fields for future use
titles_data = [(hit["pub_id"], hit["title"]) for hit in response_hits]
# Post Process Code
# Convert to DataFrame for ease of handling
titles_df = pd.DataFrame(titles_data, columns=["pub_id", "title"])

# Step 3: Perform the inner join between the two DataFrames on `pub_id`
result_df = pd.merge(titles_df, publishers_df, on="pub_id", how="inner")

# Step 4: Convert the result into a list of tuples format
result = [(row["title"], row["pub_name"]) for _, row in result_df.iterrows()]
```

Figure A2: Example of BirdES Text-to-ES

Figures A1 and A2 illustrate common queries for Single-Index and Cross-Index queries, respectively. The paradigm we propose is to convert users' natural language queries, which express their intent, into DSL and Post Process Code.

A.2 LED

A.2.1 Template Construction

```

SYSTEM:Your task is to help me construct a query template for elasticsearch based on the information related to the elasticsearch index.
The template consists of two parts, one part is the question template composed of natural language and the other part is the es query template composed of DSL+python.
My ultimate goal is to construct the real data by filling the template with values.
When a DSL query is executed using python code, the result of the query is stored into the 'response' variable, which is then parsed using the tuple list [(),()...] to parse out the content mentioned in the question from the response.
Please generate as many templates as possible, with the requirement to cover as many types of DSL queries as possible.
The text type field can use (interval, match, prefix, match_phrase, match_phrase_prefix, combined_fields, multi_match, highlight, match_all,dis_max) for these types of queries.
The geo_point type field can be queried using (geo_bounding_box, geo_distance, geo_grid, geo_polygon, geo_shape).
The join type field must be queried using (has_parent, has_child).
A field of type nested must be queried using (nested).
vector type field can use (knn) query instead of using script.
date type field can use (script, range) query.
float type field can use (max, min, sum, value_count, cardinality, avg) aggregation query.
There are also common (exists, fuzzy, regexp, term, terms, wildcard) queries.
Use as many query conditions as possible in a single DSL query statement.
The field names in the DSL query are used directly as mentioned in the original mappings, just replace the field values in the DSL query with the template.
The value of size in the DSL query should not be generated arbitrarily, but should be consistent with the quantity mentioned in the natural language question.
Generate {size} pairs of template data.
Question should be expressed in a variety of ways, rather than using a single question.
If a join type field exists, generate a query corresponding to the has_child and has_parent types.

# index name:
{index_name}

# index data summary:
{index_introduction}

# index mappings information:
```

```
{index_mappings}

# index field description:
{index_field_descritpion}

# An example of index fields value:
{index_field_value}

# Please refer to the example given to you below for your output format
[START]
question_template : "Locate and retrieve all meteorites whose mass falls within the range of [template_min_mass] to [template_max_mass] , belong to the category [template_category], and are located within a circular region centered at the coordinates (template_lat, template_lon) with a radius of [template_X] kilometers.",
query_template:‘‘‘py\nquery="{{\n  "query": {{\n    "bool": {{\n        "must": [\n        {{\n          "range": {{\n            "mass": {{\n              "gte": template_min_mass,\n              "lte": template_max_mass\n            }}\n          }}\n        }},\n  {{\n          "match": {{\n            "class": template_category\n          }}\n        }}\n      ],\n      "filter": {{\n        "geo_shape": {{\n          "geolocation": {{\n            "shape": {{\n              "type": "circle",\n              "coordinates": [template_lon, template_lat],\n              "radius": "template_Xkm"\n            }},\n            "relation": "within"\n          }}\n        }}\n    }}\n  }},\n  "_source": ["geolocation"]\n}}\nindex_name="nasa_meteorites"\nresponse = esclient.search(index=index_name,body=query)\nresult = [(hit[\'_source\'][\'geolocation\'],) for hit in response[\'hits\'][\'hits\']]"  ‘‘‘
[END]

# Your answer goes here:
[START]
question_template:
query_template:
[END]
```

The prompt used to construct templates of LED dataset

A.2.2 Question Rewrite

```

instruction: Please rewrite the following question while maintaining its original semantics. The rewritten question should convey the same core information but explore different sentence structures, such as statements, questions, commands, or other creative formats. Avoid starting with fixed formats, and try to employ diverse sentence structures and expressions. Aim to enhance the diversity of expressions while ensuring the questionâĂŹs intent and clarity remain intact, and strive for a unique and creative phrasing.
```

```


Original question: {question}
Rewritten question:

Example:
Original question: Retrieve all cities located in Latvia within the Europe region that have a latitude greater than -4.620975614106303 and less than 88.99647265103685.
Rewritten question:
# Here are some examples of rewriting given as references. Based on these examples, create more ways of rewriting divergently, but you only need to reply with one rewriting answer.
1. I would like to know which cities in Latvia, Europe, have latitudes between -4.620975614106303 and 88.99647265103685.
2. List every city in Latvia, part of the Europe region, whose latitude falls between -4.620975614106303 and 88.99647265103685.
3. In european region, which cities locate in Latvian region with latitudes ranging from -4.620975614106303 to 88.99647265103685?

Note:
1. Do not add any other extra information.
2. Your answer only needs to provide a rewritten question, do not reply with any additional information.
```

Prompt for rewriting the questions in the LED dataset

A.3 BirdES

A.3.1 Single index

```


Instruction: Convert the SQL query statement into a complete Elasticsearch query based on the Python client.The Elasticsearch's version is 8.11.2

Constraints:
1. Prohibit the use of bucket scripts

2.There is no need to define the Elasticsearch client in the generated code as the Elasticsearch client "esclient" is already provided, use "esclient" directly.

3.The index name of Elasticsearch is lowercase of the SQL table name, using underline '_' to replace space ' ' in table name.

4.The final result should be stored in the "result" variable without printing it.

5.The equality sign in SQL is equivalent to an exact match (use field.keyword) in query.

```

```
6. If the return result of SQL involves addition, subtraction,
   multiplication, and division operations, please implement it in
   Python code.

7. When calculating the total quantity using count (*), use "field":
   "_index" instead of "_id"

8.Use script when judgment logic occurs.

9.When encountering nested queries, convert step by step.

10.Using the SCAN function for querying instead of search function,
   the scan function has been declared and can be used directly.


##Examples:
sql:
```sql
{SQL_Example}
```

ES:
```py
{ES_Example}
```

Please provide the above information to convert this SQL into a query
    using Elasticsearch+Code.Don't generate other content.
SQL:{sql}

ES:
```

Prompt for transforming single table data from the Bird dataset into the BirdES dataset

A.3.2 Multiple indexes

```

### Task Description:
You need to convert the following MySQL query into an equivalent
   Elasticsearch query and implement it using the Python client.
   During the conversion process, consider the data model, query
   conditions, and result processing to ensure the final results are
   consistent with the behavior of the MySQL query.
### MySQL Query:
{sql}
### Data Model
{data_model}
### Conversion Steps:
Please follow these steps to complete the conversion:

1. **Parse the MySQL Query**:
    - Extract SELECT, FROM, JOIN, WHERE, GROUP BY, HAVING, ORDER BY,
       LIMIT clauses from the MySQL query.
```

```
    - Identify the tables involved and their join conditions.

2. **Construct the Elasticsearch Query**:
    - Build the equivalent Elasticsearch query JSON structure based on the MySQL query clauses.
    - Consider using the scan API to handle pagination and large data sets.

3. **Implement the Python Client Code**:
    - Use the elasticsearch Python client to construct and execute the query, and process the query results.
    - Firstly, use the ES query statement to filter out documents that meet the where criteria, and then use the Pandas library method to convert the ES query results to DataFrame format and perform an inner join operation. Then, use the methods provided by the Pandas library to implement group by and order by operations in SQL statements, and finally store the results in a list tuple data structure, with each tuple representing a row of data that meets the criteria.
    - Handle possible null values (None) and duplicate values to ensure the final result is consistent with the MySQL query.
### Example:
#### MySQL Query:
sql:
```sql
{SQL_Example}
```

Data Model
Data Model:
```data model
**Index Name 1**:`public_review_platform_days`
{{
    "mappings": {{
        "properties": {{
            "day_id": {{
                "type": "long"
            }}
        }}
    }}
}}
...
```
Elasticsearch Implementation in Python:
{ES_Example}
Notes
1. Ensure to handle null values (None) and duplicate values to avoid calculation errors.
...
Your Answer:
```
Prompt for transforming multi-table data from the Bird dataset into the BirdES dataset

## A.4 Human Annotation

We hired 25 annotators, including 5 ES experts and 20 students who are familiar with ES. The annotations mentioned in the article are first performed by the students, and then the experts check whether the annotation accuracy reaches 90%. If it does not reach 90%, the students continue annotating. The total cost for the annotations was $5,000.

## A.5 Sql-to-ES

SQL:

```
SELECT AVG(list_followers) FROM lists WHERE list_movie_number > 200
```

ES:

```
# Query DSL
query = {
    "query": {
        "range": {
            "list_movie_number": {
                "gt": 200
            }
        }
    },
    "aggs":{
        "avg_list":{
            "avg":{
                "field":"list_followers"
            }
        }
    }
}
#Post Process Code
index_name = "lists"
response = esclient.search(index=index_name,body=query)
result=(response['aggregations']['avg_list']['value'],)
```

SQL:

```
SELECT T2.user_trialist FROM lists AS T1 INNER JOIN lists_users AS T2 ON T1.list_id =
T2.list_id AND T1.user_id = T2.user_id WHERE T1.list_title = '250 Favourite Films'
```

ES:

```
# Step 1: Query the `movie_platform_lists` index to get documents where `list_title` is '250 Favourite
Films'
query_movie_platform_lists = {"query": {"term": {"list_title.keyword": "250 Favourite Films"}}\
}
response_hits_lists = [doc["_source"] for doc in scan(client=esclient, index="movie_platform_lists",
body=query_movie_platform_lists, size=10000)]

# Extract necessary fields for further processing
data_lists = [(hit["list_id"], hit["user_id"]) for hit in response_hits_lists]

# Step 2: Convert the extracted data into DataFrame format
df_lists = pd.DataFrame(data_lists, columns=["T1_list_id", "T1_user_id"])

# Step 3: Query the `movie_platform_lists_users` index to get all documents
query_movie_platform_lists_users = {
    "query": {
        "match_all": {}}
}
response_hits_lists_users = [
doc["_source"] for doc in scan(client=esclient, index="movie_platform_lists_users",
body=query_movie_platform_lists_users, size=10000)]
# Extract necessary fields for further processing
data_lists_users = [(hit["list_id"], hit["user_id"], hit["user_trialist"]) for hit in response_hits_lists_users]

# Step 4: Convert the extracted data into DataFrame format
df_lists_users = pd.DataFrame(data_lists_users, columns=["T2_list_id", "T2_user_id", "user_trialist"])

# Step 5: Use Pandas to perform the inner join on DataFrames
joined_data = pd.merge(df_lists, df_lists_users, left_on=
["T1_list_id", "T1_user_id"], right_on=["T2_list_id", "T2_user_id"], how="inner")

# Step 6: Extract the 'user_trialist' field from the joined data and store it in the result list of tuples
result = [(row["user_trialist"],) for index, row in joined_data.iterrows()]
```

Figure A3: Example of transform SQL into ES

Figure A3 shows an example of converting SQL to ES. On the left side, the single table query's SQL WHERE conditions are mapped to the "query" section of the DSL query, while AVG(list_followers) is mapped to the "aggregation" section of the DSL query.On the right side, multiple table query are converted into ES queries in a Cross-index Query scenario. The two SQL tables, "lists" and "lists_user," correspond to the two ES indexes, "lists" and "lists_user."The WHERE conditions from the original SQL are used to query both indexes separately, and then the JOIN conditions from the original SQL are applied to perform an Index_JOIN using the pandas merge method, ultimately returning the results.

A.6 Other Notes

We strictly adhered to the usage guidelines of the Bird (Li et al., 2024b) dataset while constructing BirdES, and the dataset we created does not contain any offensive content.

B Supplement of Evaluation Metrics

B.1 DSLEM

$$\mathbb{1}(Q_n, \hat{Q}_n) = \begin{cases} 1, & Q_n = \hat{Q}_n \\ 0, & Q_n \neq \hat{Q}_n \end{cases}$$

If Q_n exactly matches $\hat{Q}_n$, the result is assigned a value of 1; otherwise, it is assigned a value of 0.

B.2 EX

$$\mathbb{1}(O_n, \hat{O}_n) = \begin{cases} 1, & O_n = \hat{O}_n \\ 0, & O_n \neq \hat{O}_n \end{cases}$$

If O_n and $\hat{O}_n$ are exactly equal, the function $\mathbb{1}$(Âů) returns 1; otherwise, it returns 0.

B.3 VES

$$\mathbb{1}(V_n, \hat{V}_n) = \begin{cases} 1, & V_n = \hat{V}_n \\ 0, & V_n \neq \hat{V}_n \end{cases}$$

If V_n and $\hat{V}_n$ are equal, the function $\mathbb{1}$(Âů) returns 1; otherwise, it returns 0. R(Âů) is defined as follows:

$$R\left(Y_n, \hat{Y}_n\right) = \sqrt{\frac{E\left(Y_n\right)}{E\left(\hat{Y}_n\right)}}$$

E(Âů) is a metric for calculating the execution time of ES queries. By comparing the actual execution time of an ES query with the time taken to generate the ES query, we determine whether the model-generated ES queries are more efficient.

C Experimental Prompt

C.1 zero-shot

```
SYSTEM:Please write the necessary Elasticsearch query and Python code based on the given question and
Elasticsearch index mapping information. Ensure that the syntax is correct and that the query fulfills the
question's requirements and can be executed. The esclient=Elasticsearch() has already been defined, so
there is no need to define it again; use esclient directly for querying. I'll provide the relevant mapping
information in markdown format, where is_keyword denotes if a field is defined as a keyword type as well.
Generate only code and nothing else.Code format sample:
```py
#Elasticsearch DSL query
query={{}}

#Use search or scan to get data
response = esclient._()

#Store the final query execution result in tuple list format in the result variable
result =[(),]
```

INDICES DESCRIPTION:
{indices_desc}
QUESTION:{question}
ANSWER:
```

Figure A4: Prompt of zero-shot for LED, BirdES

C.2 few-shot

```
SYSTEM:Please write the necessary Elasticsearch query and Python code based on the given question and
Elasticsearch index mapping information. Ensure that the syntax is correct and that the query fulfills the question's
requirements and can be executed. The esclient=Elasticsearch() has already been defined, so there is no need to
define it again; use esclient directly for querying. I'll provide the relevant mapping information in markdown format,
where is_keyword denotes if a field is defined as a keyword type as well. Code format sample:
```py
#Elasticsearch DSL query
query={{}}
#Use search or scan to get data
response = esclient._()

#Store the final query execution result in tuple list format in the result variable
result =[(),]
```
Examples:
INDICES DESCRIPTION:
{indices_desc1}
QUESTION:{question1}
ANSWER:{answer1}

INDICES DESCRIPTION:
{indices_desc}
QUESTION:{question}
ANSWER:
```

Figure A5: Prompt of few-shot for LED, BirdES

Figure A4 and Figure A2 are the prompt templates used in our experiments, where indices_desc refers to the description information of the indexes.

D Supplement of Experiment

D.1 zero-shot results

Models	LED			BirdES		
	DSL-EM	EX	VES	DSL-EM	EX	VES
Advanced Models						
LLaMA2-7B	0	0	0	0	0	0
LLaMA2-7B-Chat	0	0	0	0	0	0
LLaMA2-13B	0	0	0	0	0	0
LLaMA2-13B-Chat	0	0	0	0	0	0
LLaMA3-8B-Instruct	0	0	0	0	0	0
LLaMA3.1-8B-Instruct	0.06	0.12	0.13	0	0.07	0.13
Qwen1.5-7B	0	0	0	0	0	0
Qwen1.5-7B-Chat	0	0	0	0	0	0
Qwen2.5-Coder-7B-Instrcut	0	2.72	2.93	0	0	0
GPT4o	**3.15**	**25.05**	**26.08**	**0.98**	**11.93**	**18.69**
Code Models						
CodeLLaMA-7B	0	0	0	0	0.13	0.41
CodeLLaMA-7B-Instruct	0	0	0	0.06	0.13	<u>0.43</u>
CodeLLaMA-13B	0	0.06	0.15	0	0	0
CodeLLaMA-13B-Instruct	0	0	0	0	0	0
CodeQwen1.5-7B	0	0	0	0	<u>0.52</u>	1.15
CodeQwen1.5-7B-Chat	0	0.06	0.06	0	0	0
Deepseek-Coder-6.7B-Base	0	0	0	0	0	0
Deepseek-Coder-6.7B-Instruct	0	0	0	0	0	0
Ours						
FeynMan-13B	<u>1.48</u>	<u>11.26</u>	<u>11.75</u>	<u>0.13</u>	0.13	0.16

Table A1: Performance of zero-shot on LED and BirdES

Compared to the base model, our trained FeynMan-13B model improved by 11.26% on the LED dataset and also showed slight improvements on the BirdES dataset, demonstrating the effectiveness of our dataset. It can be observed that almost all models perform very poorly in a zero-shot setting, except for GPT-4o, which achieves some level of effectiveness.Therefore, in the main experiments presented in the text, we adopted a few-shot setting.It is evident that for the new task of Text-to-ES, most models currently perform poorly.

D.2 Training Details and Hyper-parameters

The fine-tuning of the FeynMan-13B model, which is based on CodeLLaMA-13B, was conducted on both the LED and BIRDES datasets, resulting in respective FeynMan-13B models for each dataset, using 8 A100 GPUs. LoRA was applied to the query and value projection matrices, with a per-device batch size of 4, gradient accumulation steps of 4, and a learning rate of 5×10^{-5}, utilizing a cosine scheduler. Training spanned 4 epochs with mixed precision (fp16) enabled.

E Elasticsearch VS MySQL

E.1 Feature Comparison: ES vs SQL

Elasticsearch differs from traditional relational databases (RDBMS) in several key ways, as illustrated in Table A2. In Elasticsearch, data is stored in indexes, whereas SQL databases organize data in tables.An index in Elasticsearch is equivalent to a table in SQL, and queries can only be directed to a single index. Instead of using key-value pairs, Elasticsearch stores documents in JavaScript Object Notation (JSON) format, which means that query statements are also expressed in JSON Query Domain Specific Language. Additionally, Elasticsearch is schema-free, allowing two documents within the same index to have different schemas, while rows in an RDBMS must adhere to an identical schema.

Elasticsearch element	SQL element
Index	Database
Mapping	Schema
Document type	Table
Document	Row
Schema-Free	Schema-Fixed

Table A2: Features Comparison between Elasticsearch and MySQL

E.2 Efficiency Analysis: ES vs SQL

Our experimental setup includes equivalent queries Q_sql and Q_dsl for SQL and DSL. We measured the average execution time of the queries executed three times on the MySQL single table Table_mysql and the Elasticsearch single index Index_es with the same scale of data. We inserted the original size of data each time for T_mysql and T_es. We recorded the trend of SQL query time T_sql and ES query time T_es as the data scale increases linearly.As shown in Figure A6, initially, the execution time of ES was higher than that of SQL. However, as the data increased, the execution time of SQL increased linearly, while the execution time of ES increased logarithmically. Eventually, after the number of returned documents reached our set limit of 5000, both reached a stable trend.

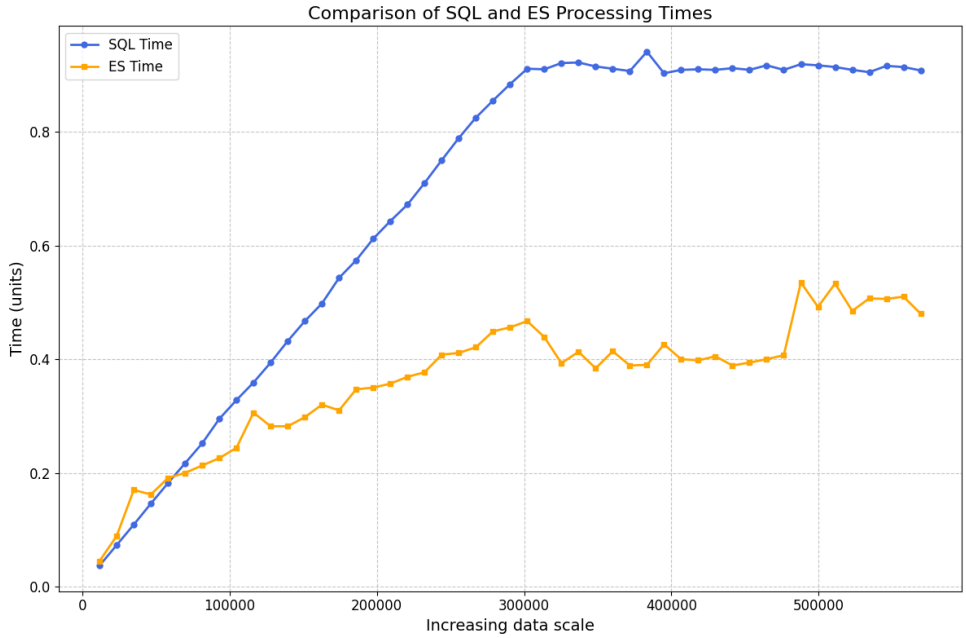

Figure A6: Query time trend chart for SQL and ES with equivalent query statements as data scale grows