Text-to-ES Bench: A Comprehensive Benchmark for Converting Natural Language to Elasticsearch Query

Anonymous ACL submission

Abstract

Elasticsearch (ES) is a distributed RESTful search engine optimized for large-scale and long-text search scenarios. Recent research on Text-to-Query has explored using large language models (LLMs) to convert user query intent to executable code, making it an increasingly popular research topic. To our knowledge, we are the first to introduce the novel semantic parsing task text-to-ES. To bridge the gap between LLM and ES, in detail, we leverage LLMs to generate Domain-Specific Language (DSL) and corresponding post-processing code to support multi-index ES query. Consequently, we propose the text-to-ES benchmark that consists of two datasets: **L**arge **E**lasticsearch **D**ataset (**LED**), containing 26,207 text-ES pairs derived from a 224.9GB schema-free database, and **E**lastic**S**earch (**BirdES**)with 10,926 pairs sourced from the **Bird** dataset on a 33.4GB schema-fixed database. Compared with ten advanced LLMs and six code-based LLMs, the model we trained outperformed GPT-4o by 21.79% on the LED dataset, setting a new state-of-the-art, and achieved 98% of GPT-4o's performance on the BirdES dataset. Additionally, we provide in-depth experimental analyses and suggest future research directions for this task. We will release our code and datasets in the future.

1 Introduction

Elasticsearch (ES) is a distributed RESTful search engine (Akdal et al., 2018b) that offers powerful full-text search capabilities and supports schema-free scenarios, allowing it to process petabytes of data in seconds[1]. At present, people can only interact with the ES database by manually writing ES query, which presents several challenges. (1) **Using wrong keywords**. For instance, in the green part of Figure 1, it is difficult to organize appropriate

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

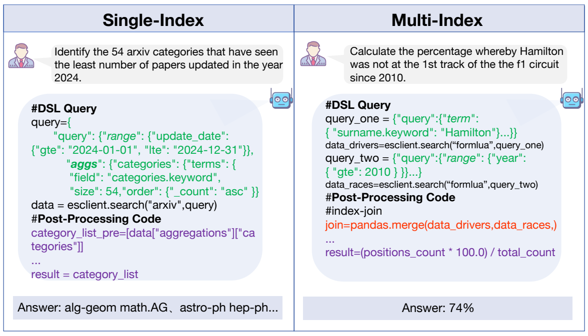

Figure 1: Example of converting natural language to ES query statement. Left: Query involves a single index. Right: Query involves multiple indexes. The green part denotes the Domain-Specific Language query body, the red part denotes the index join option, and the purple part denotes the post-processing code.

ES keywords to express information from natural language, such as aggregation information. (2) **Index join error**. In the red part of Figure 1 right where a natural language question involves multiple indexes, assessing the logic of index joining is quite Challenging. (3) **Generating wrong post-processing code**. In the purple part of Figure 1, both single-index and multi-index query require writing appropriate post-processing code, which is complex.

Text-to-Query refers to the process of utilizing large language models (LLMs) to automatically translate user intent into executable code, which can alleviate the three challenges faced by ES. Currently, the most rapidly developing area is text-to-SQL (Zhong et al., 2017; Yu et al., 2018; Li et al., 2024b), which transforms natural language into SQL query. 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-OverpassQL (Staniek et al., 2024), text-to-CQL (Lu et al., 2024) and text-to-SPARQL (Yin et al., 2021). However, there is a lack of research on the automatic generation of

Dataset	# Size	# Row/Index	# Column/Index	# Scale	Domain	Schema-Free
WikiSQL	81,654	0.01k	6	0.2GB	SQL	✗
Spider	10,181	1k	5	1.7GB	SQL	✗
Bird	12,751	530k	4	33.4GB	SQL	✗
BirdES(ours)	10,962	530k	4	33.4GB	ES	✗
LED(ours)	26,207	88k	37	224.9GB	ES	✔

Table 1: Comparison of text-to-SQL datasets. Size represents the number of datasets. Row/Index indicates the average number of data rows per index, while Column/Index denotes the average number of columns per index, with LED reaching a maximum value of 37. Scale refers to the corresponding database size of the dataset, with LED achieving an enormous size of 224.9 GB. Domain represents the query statements used in the dataset. Schema-Free indicates the flexibility of the dataset; in LED, the schema of any two rows can differ, whereas in SQL, the schema of any two rows must remain consistent. For more schema-free details, see the Appendix A12.

queries for ES.

In this paper, we explore the text-to-ES task and evaluate the performance of LLMs. To our knowledge, we are the first to propose this task, a novel semantic parsing problem well-motivated in real-world applications. The task aims to convert natural language to ES query. To bridge the gap between LLMs and ES, we leverage LLMs to generate Domain-Specific Language (DSL) and corresponding post-processing code, enabling ES to support multi-index query, as illustrated in Figure 1 right. Based on the text-to-ES task, we propose the text-to-ES benchmark that consists of two datasets. To address the challenges of writing ES query, we collected data from Wikipedia and Kaggle to create **LED**, a **L**arge-scale **ES** **D**ataset grounded in text-to-ES, containing 26,207 text-to-ES pairs with a total size of 224.9 GB. In this manner, we constructed the **Bird** **E**lastic**S**earch (**BirdES**) dataset, derived from the Bird (Li et al., 2024b) dataset in the text-to-SQL domain. The BirdES dataset consists of 10,962 text-to-ES pairs, with nearly 80% of the data representing multi-index query and featuring a highly complex index structure. The comparison table with text-to-SQL is shown in Table 1.

Ultimately, we conduct extensive experiments using ten advanced models and six code models on our LED and BirdES datasets. The model we trained outperformed GPT-4o (OpenAI, 2024) by 21.79% on the LED and achieved 98% of GPT-4o's performance on the BirdES. We also performed manual sampling evaluations on our datasets, achieving scores of 95% and 99%, respectively. In addition, we suggest future research directions for this task. We believe that our work will contribute to advancing real-world applications of text-to-ES research. Our contribution is as follows.

- To our knowledge, we are the first to propose a semantic parsing task text-to-ES. To bridge the gap between LLMs and ES, we leverage LLMs to generate DSL and post-processing code to support multi-index ES query.
- We propose the large text-to-ES benchmark consists of two datasets, LED and BirdES. LED has 26,207 Text-ES pairs with a 224.9 GB schema-free database, and BirdES has 10,962 Text-ES pairs with a 33.4 GB schema-fixed database.
- We conduct extensive evaluation and analysis experiments using ten advanced and six code LLMs. The model we trained outperformed GPT-4o by 21.79% on the LED and achieved 98% of GPT-4o's performance on the BirdES. Additionally, we perform manual sampling assessments on our datasets.

2 Releated Work

2.1 Text-to-Query

Text-to-Query is the process of using LLM to convert user intent into 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). The second category 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

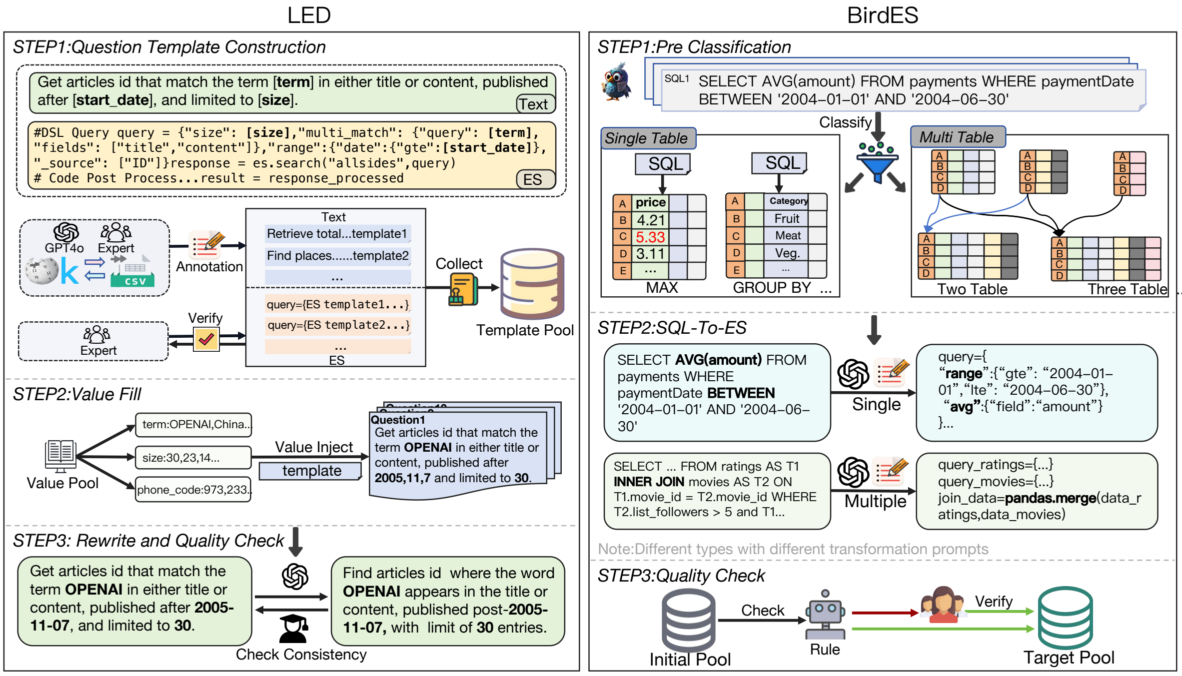

Figure 2: Detailed flowchart of data construction. On the left, the LED data construction process is depicted, where Text-ES template pairs are created using multiple experts in collaboration with GPT-4o. The templates are populated with values from the database, and the constructed data is rewritten using GPT-4o. 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 query to build BirdES, which is subsequently evaluated.

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 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, which bridge the gap in this domain in terms of automatic querying of ES database.

2.2 Domain-Specific 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 ES query. At present, LLMs excel in generating code for languages like Python. 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 query with LLMs, especially for non-standard experimental constraints. Although Akdal et al. (2018a) explore integrating heuristic rules to generate ES query, we propose an advanced LLM-based text-to-ES task, which serves as a more standardized approach for the automated generation of ES query.

3 Text-to-ES Task Formulation

Text-to-ES refers to the process of converting a natural language question $\mathcal{Q}$ into an ES query $\mathbf{E}$ capable of retrieving relevant data from ES database. The schema information can be represented as $\mathcal{S} = \langle \mathcal{F}, \mathcal{I} \rangle$, where $\mathcal{F}$ and $\mathcal{I}$ are fields and indexes respectively. Finally, the text-to-ES could be formulated as:

$$\begin{aligned} \mathcal{D}, \mathcal{C} &= f(\mathcal{Q}, \mathcal{S} \mid \boldsymbol{\theta}), \\ \mathbf{E} &= \mathcal{C}(\mathcal{D}), \end{aligned} \tag{1}$$

where the function $f(\cdot \mid \boldsymbol{\theta})$ represents a model with parameters $\boldsymbol{\theta}$, $\mathcal{D}$ represents DSL and $\mathcal{C}$ represents post-processing code. The post-processing code assists Elasticsearch in executing multi-index queries.

4 Data Construction

4.1 LED

The LED dataset encompasses nearly all commonly used DSL from the official ES documentation[2].

4.1.1 Template Construction

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 ES documentation. We collected a substantial amount of long-text data from open data platforms such as Kaggle[3] and PaperWithCode[4]. Additionally, we incorporated geographic data to leverage ES's geographic query capabilities. After collection, we engaged three ES experts to collaborate with GPT-4o in constructing 2,783 text-ES template pairs, as illustrated in the left of Figure 2. Ultimately, approximately 2,600 text-ES template pairs were constructed.

4.1.2 Value Filling

The value-filling step inserts appropriate values into the created templates to form text-ES pairs. We employed both automated data retrieval and manual input methods. We directly extract relevant data from the ES index for non-open fields, such as names and geographical locations. For open fields, such as title keywords, the values are manually crafted based on the ES index data. Through the two approaches, we develop a Value Pool. By inputting a text-ES template into the Value Pool, we generate approximately ten text-ES pairs. For the well-filled data, we execute each text-ES pair one by one. If the execution fails or the result is empty, we carefully revise that data.

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, ultimately improving the overall quality and diversity of the generated outputs.

[2]https://www.elastic.co/query-dsl.html
[3]https://www.kaggle.com
[4]https://paperswithcode.com

4.1.4 Quality Control

In the quality control 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 98% of the samples meet both completeness and readability standards, it indicates that the dataset quality has passed inspection.

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 multi-index query.

4.2.1 Pre-Classification

The pre-classification step is designed to categorize the SQL data into different classes. We initially classify the queries into single-table and multi-table based on the number of tables involved in the SQL statements. Furthermore, we categorize single-table query by keywords, dividing them into categories such as MAX, LIKE, and GROUP BY and so on. In contrast, multi-table query are classified based on the number of tables involved, such as two-table, three-table, and so on. We use different transformation methods for SQL data of different categories.

4.2.2 Single Table Conversion

We employed a human-machine collaboration approach to transform 2,610 single-table SQL query into corresponding single-index ES query. In detail, the SQL 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. For specific examples see Appendix A7

4.2.3 Multiple Table Conversion

We also utilized a human-machine collaboration approach to convert multi-table data. In detail, we first used post-processing code to address the challenge of ES not supporting multi-index query. We carefully constructed SQL-to-ES examples as in-context learning (ICL) for GPT-4o, allowing it to perform an initial transformation on 7,212 records. For any data that did not pass the transformation, we manually adjusted it one by one. Additionally, we meticulously transformed 1,140 multi-index records in the test set manually. It is worth mentioning that we attempted to overcome the limitation of multi-index query by converting them into multiple single-index query. However, we found this approach unfeasible, and Zhang et al. (2023a) faced the same constraint.

4.2.4 Quality Control

The quality control step focuses on primarily verifying whether the execution results of the original SQL and ES are consistent. It is important to note that discrepancies between the execution results of ES and SQL do not necessarily indicate that ES is incorrect. For example, when multiple records meet the query conditions but only one record is required, the returned results from SQL and ES are likely to differ, yet the ES query can still be correct. 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 90%.

4.3 Data Statistics

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. The total size of the BirdES index data is 33.4 GB, containing 10,962 Text-ES pairs. Nearly 80% of the data consists of multi-index query. The BirdES dataset includes 9,428 train samples and 1,534 test samples.

Dataset	Train	Dev	Test	Total
BirdES	9,428	-	1,534	10,962
LED	23,099	1,569	1,539	26,207

Table 2: Statistics of our constructed BirdES and LED.

5 Experiment

5.1 Experiment Models

We select 16 representative LLMs covering 10 advanced models and 6 code models as follows:

Advanced Model We used the LLaMA series models (Touvron et al., 2023) includes {LLaMA2-7b-Chat, LLaMA2-13b-Chat, LLaMA2-70B-Chat, LLaMA3-8B-Instruct, and LLaMA3.1-8b} and the Qwen 2.5-Instruct series models (Yang et al., 2024) covering {7B, 14B, 32B, 72B}.

Code Model We utilized the CodeLLaMA-Instruct series models {7B, 13B, 34B} (Roziere et al., 2023) and the Qwen2.5-Coder-Instruct series models {7B, 14B, 32B} (Hui et al., 2024).

Fine-tuning Model Additionally, we chose Qwen2.5-Coder-14B-Instruct as our base model and trained two distinct models using the train set of LED and BirdES datasets, respectively.

Human Evaluation We further designed a manual answering method. In this approach, we randomly selected 100 samples from both LED and BirdES and invited three undergraduate students (different from the data construction team in Section 4) to answer the questions. The evaluation method is consistent with the evaluation of the model inference results.

5.2 Experiment Setup

In this section, we clarify the evaluation metrics and implementation details.

5.2.1 Evaluation

Domain-Specific 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:

$$\text{DSLEM} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(Q_n, \hat{Q}_n) \tag{2}$$

The ground truth DSL is represented as Q_n and the generated DSL as $\hat{Q}_n$. If Q_n exactly matches $\hat{Q}_n$,

Models	LED			BirdES		
	DSL-EM	EX	VES	DSL-EM	EX	VES
Advanced Models						
LLaMA3-8B-Instruct (one-shot)	3.37	6.04	5.85	0.07	0.15	0.15
LLaMA3.1-8B-Instruct (one-shot)	10.66	16.89	17.26	0.59	1.17	1.29
LLaMA2-7B-Chat (one-shot)	4.48	5.07	5.06	0	0.22	0.22
LLaMA2-13B-Chat (one-shot)	13.84	24.37	24.51	0	0.39	0.72
LLaMA2-70B-Chat (one-shot)	21.64	26.97	27.45	0.29	0.81	0.74
Qwen2.5-7B-Instruct (one-shot)	0.51	7.01	7.27	1.24	9.61	12.04
Qwen2.5-14B-Instruct (one-shot)	2.14	15.91	16.25	1.62	16.96	19.68
Qwen2.5-32B-Instruct (zero-shot)	0	9.74	10.88	0	3.67	4.53
Qwen2.5-32B-Instruct (one-shot)	8.12	27.55	29.11	2.05	23.34	26.74
Qwen2.5-72B-Instruct (zero-shot)	0.83	17.86	20.14	0.29	12.33	17.68
Qwen2.5-72B-Instruct (one-shot)	27.95	43.28	43.86	2.93	25.03	26.74
GPT4o (zero-shot)	3.15	25.05	26.08	0.98	11.93	18.69
GPT-4o (one-shot)	30.15	48.73	49.42	2.35	**25.75**	**35.12**
Code Models						
CodeLLaMA-7B-Instruct (one-shot)	0.37	5.79	6.20	0	0.39	0.72
CodeLLaMA-13B-Instruct (one-shot)	19.42	26.19	26.20	0.13	0.52	0.43
CodeLLaMA-34B-Instruct (one-shot)	33.91	43.92	44.81	0.81	4.63	4.90
Qwen2.5-Coder-7B-Instruct (one-shot)	2.46	11.24	11.89	1.91	12.99	13.36
Qwen2.5-Coder-14B-Instruct (one-shot)	2.22	19.49	21.64	3.30	22.32	22.96
Qwen2.5-Coder-32B-Instruct (zero-shot)	0.38	15.72	17.83	0	4.91	5.41
Qwen2.5-Coder-32B-Instruct (one-shot)	26.57	43.79	44.84	<u>3.34</u>	24.81	<u>28.16</u>
Fine-tuning						
Qwen2.5-14B-Coder-FeynMan (zero-shot)	6.88	23.52	24.19	1.54	4.47	3.68
Qwen2.5-14B-Coder-FeynMan (one-shot)	**48.27**	**62.31**	**63.25**	**4.04**	<u>25.25</u>	23.19
Human Evaluation						
Human (sampleing)	81.00	95.00	97.29	83.00	99.00	99.56

Table 3: DSLEM denotes Domain-Specific Language Exact Match Accuracy. EX denotes Execution Accuracy. VES denotes Valid Efficiency Score Performance comparison on LED and BirdES benchmarks. The best results are highlighted in **bold**. The second results are highlighted by <u>underline</u>. All zero-shot experimental results can be found in the Appendix A1.

the function $\mathbb{1}(\cdot)$ 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 generated query and the ground truth result. The formula is shown below:

$$\text{EX} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}(O_n, \hat{O}_n) \tag{3}$$

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 $\mathbb{1}(\cdot)$ 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:

$$\text{VES} = \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}(V_n, \hat{V}_n)\cdot\mathbf{R}(Y_n, \hat{Y}_n) \tag{4}$$

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 $\mathbb{1}(\cdot)$ is used to determine whether V_n and $\hat{V}_n$ are equal. $\mathbf{R}(\cdot)$ is a function that evaluates the efficiency ratio. Further details can be found in the derivation of formulas section of Appendix B.3.

5.2.2 Implementation details

Zero-shot results can be found in Appendix A1. To ensure the stability of the experiments, the temperature for all models used was set to 0.0001, and

Models	Specialized		TermLevel		FullText		Geography		Joining		Aggregation	
	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX	DSLEM	EX
GPT-4o (zero-shot)	0.00	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
CodeLLaMA-34B-Instruct	36.23	36.23	31.33	46.00	21.60	28.00	41.33	48.00	34.00	44.00	35.33	49.33
Qwen2.5-32B-Coder-Instruct	10.14	27.54	27.33	45.33	13.60	35.20	50.00	60.00	30.67	36.00	15.33	48.67
Qwen2.5-14B-Coder-Instruct (one-shot)	4.35	10.14	2.00	37.33	0	14.40	0.00	13.33	2.00	5.33	2.00	22.67
Qwen2.5-14B-Coder-FeynMan	**43.48**	46.38	**54.00**	**63.33**	**34.40**	**63.45**	**56.00**	**64.67**	**48.67**	**62.67**	**48.00**	**68.00**

Table 4: Model Performance on Different Categories in the LED Dataset.

all other hyperparameters were maintained at their default values. The model training method was LoRA (Hu et al., 2021), with learning rates and other parameters detailed in the Appendix D.2. Additionally, all experiments in the main results and analysis were conducted using a one-shot approach. For ICL selection, we utilize the llm-embedder (Zhang et al., 2024) model to select one example from the train set.

5.3 Main Results

The experimental results are presented in the Table 3. From the table, we notice that: 1) **All LLMs perform poorly on the LED and BirdES datasets**, with even GPT-4o achieving only 11.93% in a zero-shot setting. The best-performing model, Qwen2.5-Coder-FeynMan-14B, achieves accuracies of 62.31% and 25.25%, respectively. 2) **Among models of the same series, larger models tend to perform better**. This is evident from the performance of the LLaMA2, Qwen2.5, CodeLLaMA, and Qwen2.5-Coder series models shown in Table 3. 3) **Models fine-tuned with code outperform their base models**. CodeLLaMA outperforms LLaMA2 under the same parameters, and Qwen2.5-Coder models show similar results. 4) **The models fine-tuned on our dataset outperform their base models.** The fine-tuned two Qwen2.5-14B-Coder-FeynMan significantly exceed the performance of its base model Qwen2.5-Coder-14B-Instruct on both LED and BirdES. The model we trained outperformed GPT-4o by 21.79% on the LED and achieved 98% of GPT-4o's performance on the BirdES respectively. 5) **One-shot demonstration significantly improves the performance.** The models we trained improved by 62.25% on LED and 82.30% on BirdES, while GPT-4o achieved improvements of 48.59% and 53.66% on the same datasets, respectively.

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 (one-shot)	2.8	34.5	36.5	2.2	13.6	21.9
Qwen2.5-72B-Instruct (one-shot)	2.63	32.89	33.00	4.21	22.37	30.00
Qwen2.5-Coder-32B-Instruct (one-shot)	2.89	33.16	33.00	3.16	21.32	25.00
Qwen2.5-14B-Coder-Instruct (one-shot)	3.42	31.58	32.00	3.68	17.89	20.00
Qwen2.5-14B-Coder-FeynMan (zero-shot)	1.58	7.63	5.00	2.11	3.68	3.00
Qwen2.5-14B-Coder-FeynMan (one-shot)	2.89	36.84	36.00	4.47	18.68	16.00

Table 5: Performance of Different Models in Single-Index and Multi-Index Scenarios

5.4 Detailed Analysis

In this section, we focus on five problems: (1) Which types of ES query affect the performance? (2) Are multiple index ES queries more difficult than single index? (3) How do the LLMs perform at different levels of difficulty? (4) Can external knowledge improve the performance of LLM? (5) What types of errors can LLMs make in the text-to-ES benchmark?

5.4.1 Analysis for different types of ES

For the LED dataset, we categorized the questions based on the types outlined on the ES official website. For example, queries related to geography were classified as "Geography". Ultimately, we divided the dataset into six categories: "Specialized", "TermLevel", "FullText", "Geography", "Joining", and "Aggregation,". Detailed descriptions of each category can be found in Appendix D.3. As shown in Table 4, we draw the following conclusions: Compared to other models, Qwen2.5-14B-Coder-FeynMan demonstrates significant improvement across various categories. Specifically, it achieves an accuracy of approximately 63% in nearly all categories, except for "Specialized". This lower performance in the "Specialized" category may be attributed to its higher complexity, which involves not only generating basic DSL but also more intricate script code within the DSL (see Appendix A10).

5.4.2 Analysis for Single and Multiple Index

For the BirdES dataset, we classified queries based on the number of indices involved, dividing them into two categories: single-index and multiple-index. From Table 5, we observe that: 1) All models performed better on the Single-Index than on the Multi-Index, with average improvements of over 12.8% and 9.82% in EX and VES in one-shot setting, respectively. 2) One-shot demonstration is limited for Multiple-Index. For example, the average improvement of GPT-4o and Qwen-FeynMan on the single-index query is 73.11%, while it is 58.89% for the multi-index query. This highlights the challenges of multi-index in text-to-ES tasks.

5.4.3 Analysis for different levels of difficulty

We follow the prior work (Li et al., 2024b) and set three levels {simple, moderate, challenging} in BirdES aligned with the Bird dataset. The results of different models at varying levels of difficulty are shown in Figure 3. We observe that as data complexity increases, model performance declines. This indicates that challenging SQL query continue to pose difficulties for ES query.

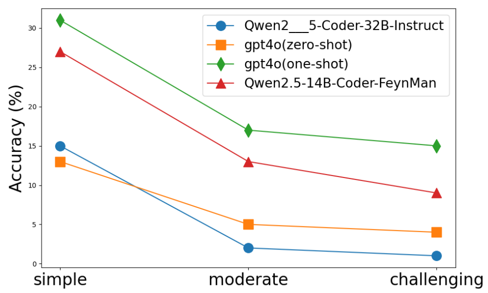

Figure 3: Trend of Model Performance with Increasing Difficulty.

5.4.4 Analysis for incorporating external knowledge

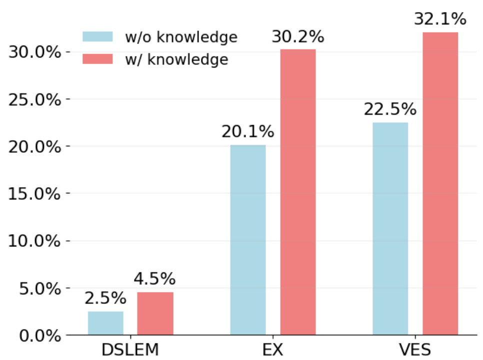

Figure 4: Average score of LLMs on BirdES.

Leveraging the external knowledge from the Bird dataset, we explore the impact on text-to-ES tasks. We used all 7 models of Qwen2.5 along with GPT-4o and two trained models in both "with knowledge" and "without knowledge" settings. The average results are presented in Figure 4. From the figure, we observe that, in the presence of knowledge, the model performs better on all three metrics compared to the absence of knowledge. Both the EX and VES improved by approximately 10%. Detailed results are in Appendix D.5.

5.4.5 Error Analysis

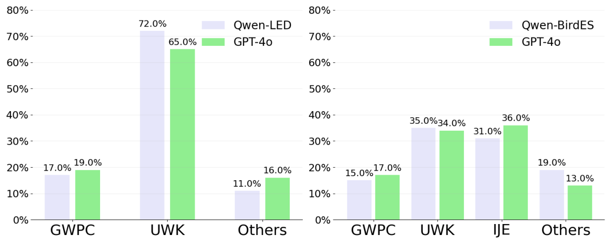

Figure 5: Distribution of error types.

To guide future research on LLMs in text-to-ES tasks, we analyze 100 error samples generated by the trained model and GPT-4o, the best LLMs on LED and BirdES. After manual review, we categorized three error types: (a) generating the wrong post-processing code(GWPC), (b) using the wrong keywords (UWK), and (c) index join error (IJE). The error distribution is shown in Figure 5. In the LED dataset, the main error type for GPT-4o and FeynMan is UWK. In the BirdES dataset, the primary error types for GPT-4o and FeynMan are IJE and UWK. Detailed examples of each category are provided in the Appendix D.6.

6 Conclusion

In this paper, we first propose the text-to-ES task and leverage large language models to generate Domain-Specific Language and post-processing code to support multi-index Elasticsearch query. Based on our constructed LED and BirdES datasets, we introduce a comprehensive text-to-ES benchmark. Additionally, we conduct extensive evaluations and analyses using ten advanced LLMs and six code-focused LLMs. Our trained model achieved outstanding results. Furthermore, we perform manual sampling assessments on our datasets. We hope that our work will contribute to advancing real-world applications of text-to-ES research.

Limitation

(1) Our dataset labeling requires collaboration between humans and GPT-4o, and we cannot fully rely on GPT-4o for automated labeling yet. (2) We explored methods to improve text-to-ES performance on models in the 14B parameter range, but we also focused on enhancement methods for smaller models, such as those in the 7B range.

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 release the datasets publicly for research purposes in the future. To 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*.

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.

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

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.

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.

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*.

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)*.

Michael Staniek, Raphael Schumann, Maike Züfle, and Stefan Riezler. 2024. Text-to-overpassql: A natural language interface for complex geodata querying of openstreetmap. *Transactions of the Association for Computational Linguistics*, 12:562–575.

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*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Zhicheng Dou, and Jian-Yun Nie. 2024. A multi-task embedder for retrieval augmented llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3537–3553.

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.
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
query="{{
  "query": {{
    "bool": {{
      "must": [
        {{
          "range": {{
            "mass": {{
              "gte": template_min_mass,
              "lte": template_max_mass
            }}
          }}
        }},
  {{
          "match": {{
            "class": template_category
          }}
        }}
      ],
      "filter": {{
        "geo_shape": {{
          "geolocation": {{
            "shape": {{
              "type": "circle",
              "coordinates": [template_lon, template_lat],
              "radius": "template_Xkm"
            }},
            "relation": "within"
          }}
        }}
      }}
    }}
  }},
  "_source": ["geolocation"]
}}
index_name="nasa_meteorites"
response = esclient.search(index=index_name,body=query)
result = [(hit['_source']['geolocation'],) for hit in response['hits']['hits']]"
```
[END]

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

Figure A3: The prompt used to construct templates of LED dataset

The prompt defines templates for the LED dataset, combining natural language questions and Elasticsearch DSL queries. It supports complex queries, for generating diverse and structured data.

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 question's 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.
```

Figure A4: Prompt for rewriting the questions in the LED dataset

This prompt is designed to rewrite questions in the LED dataset to maintain original meaning while enhancing diversity.

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

CONSTRAINT:
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.

EXAMPLE:
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:
```

Figure A5: Prompt for transforming single table data from the Bird dataset into the BirdES dataset

The prompt provides instructions and constraints for converting single-table SQL queries from the Bird dataset into Elasticsearch queries to generate 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:
```

Figure A6: Prompt for transforming multi-table data from the Bird dataset into the BirdES dataset

The prompt provides instructions and constraints for converting multiple-table SQL queries from the Bird dataset into Elasticsearch queries to generate 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

Figure A7 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.

SQL:

```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:

```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 A7: Example of transform SQL into ES

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 $\mathbb{1}(\cdot)$ 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 $\mathbb{1}(\cdot)$ returns 1; otherwise, it returns 0. $\mathbf{R}(\cdot)$ is defined as follows:

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

$\mathbf{E}(\cdot)$ 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 784

C.1 zero-shot 785

```
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 A8: Prompt of zero-shot for LED, BirdES

C.2 few-shot 786

```
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 A9: Prompt of few-shot for LED, BirdES

Figure A8 and Figure A2 are the prompt templates used in our experiments, where indices_desc refers 787
to the description information of the indexes. 788

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	0	0	0	0	0	0
LLaMA3-8B-Instruct	0	0	0	0	0	0
LLaMA3.1-8B	0	0	0	0	0	0
LLaMA3.1-8B-Instruct	0.06	0.12	0.13	0	0.07	0.13
Qwen2.5-7B-Instruct	0.13	2.92	0.19	7.530	7.66	
Qwen2.5-14B-Instruct	0	5.84	6.59	0	0.44	0.51
Qwen2.5-32B-Instruct	0	9.74	10.88	0	3.67	4.53
Qwen2.5-72B-Instruct	0.83	17.86	20.14	0.29	12.33	17.68
Qwen1.5-7B	0	0	0	0	0	0
Qwen1.5-7B-Chat	0	0	0	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	0.43
CodeLLaMA-13B	0	0.06	0.15	0	0	0
CodeLLaMA-13B-Instruct	0	0	0	0	0	0
CodeLLaMA-34B-Instruct	0	3.19	3.25	0	0	0
CodeQwen1.5-7B	0	0	0	0	0.52	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
Qwen2.5-Coder-7B-Instruct (Hui et al., 2024)	0.19	7.53	7.66	0	0	0
Qwen2.5-Coder-14B-Instruct	0	5.84	6.59	0	1.46	3.60
Qwen2.5-Coder-32B-Instruct	0.38	15.72	17.83	0	4.91	5.41
Fine-tuning						
Qwen2.5-14B-Coder-FeynMan	6.88	23.52	24.19	1.54	4.47	3.68

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

D.2 Training Details and Hyper-parameters

We fine-tuned the Qwen2.5-Coder-14B-Instruct model on the training datasets of LED and BirdES, resulting in our model Qwen2.5-Coder-14B-FeynMan. We trained on four A100 (40GB) GPUs for approximately 16 hours, with the final loss reduced to around 0.09. We trained for one epoch with a learning rate of 5×10^{-5}, utilizing a cosine scheduler.

D.3 Details types introduction

We have categorized the data into six groups based on the functionality of keywords, referencing the classification method from the official documentation, which is noted in the main text as footnote 2. **TermLevel** involves precise search categories, such as range searches and exact matches. **Fulltext** pertains to ES queries aimed at strings, commonly used for full-text search. **Geograph** focuses on ES queries related to geographic data structures, such as Geo-grid searches. **Joining** relates to ES nested and parent/child type queries. **Specialized** includes specific queries, such as using scripts in DSL for querying. Finally, **Aggregation** refers to ES queries aimed at statistical analysis, such as max and min.

D.4 Specialized example

The script type under the Specialized category allows for writing complex painless code in DSL statements, as shown in Figure A10.

```
# Step 2: Query match data that meets the criteria
match_query = {
    "query": {
        "bool": {
            "must": [
                {"terms": {"league_id": league_ids}},
                {"term": {"season.keyword": "2009/2010"}},
                {
                    "script": {
                        "script": {
                            "source": """
                                if (doc['away_team_goal'].size() > 0 && doc['away_team_goal'].value != null &&
                                    doc['home_team_goal'].size() > 0 && doc['home_team_goal'].value != null) {
                                    return doc['away_team_goal'].value - doc['home_team_goal'].value > 0;
                                } else {
                                    return false;
                                }
                            """,
                            "lang": "painless"
                        }
                    }
                }
    ] } }}
```

Figure A10: Example of Specidalized

D.5 Details for ablation

Models	without knowledge			with knowledge		
	DSLEM	EX	VES	DSLEM	EX	VES
Qwen2.5-7B-Instruct (Yang et al., 2024)	1.24	9.61	12.04	1.68	13.36	15.13
Qwen2.5-14B-Instruct (Yang et al., 2024)	1.62	16.96	19.68	2.56	22.02	24.21
Qwen2.5-32B-Instruct	2.05	23.34	26.74	3.02	33.77	38.49
Qwen2.5-72B-Instruct	2.93	25.03	26.74	5.07	35.09	38.20
Qwen2.5-Coder-7B-Instruct (Hui et al., 2024)	1.91	12.99	13.36	2.86	14.83	16.27
Qwen2.5-Coder-14B-Instruct	3.34	22.32	22.96	4.91	31.57	33.16
Qwen2.5-Coder-32B-Instruct	3.30	24.81	28.16	5.72	37.59	40.32
GPT-4o (OpenAI, 2024)	**2.35**	**25.75**	**35.12**	**7.78**	**42.80**	**44.89**
Qwen2.5-14B-Coder-FeynMan	3.67♣	20.11♣	17.67♣	7.05♠	40.61♠	37.82♠

Table A2: Performance comparison on LED and BirdES benchmarks. The best results are highlighted in **bold**. The base model is Qwen2.5-Coder-14B-Instruct. The symbol ♣ denotes training without knowledge, while ♠ indicates training with knowledge.

As illustrated in Table A3 Without the provided knowledge information, the model cannot know to use ’h’ and ’c’ to represent carbon and hydrogen.

D.6 Error examples

Type	Question	Knowledge
with knowledge	Calculate the total atoms consisting of the element carbon and hydrogen.	consisting of element carbon and hydrogen refers to element in('c', 'h')
without knowledge	Calculate the total atoms consisting of the element carbon and hydrogen.	-

Table A3: Example of with or without knowledge

Type	Question	Ground Truth	Error of LLMs
UWK	Calculate the average rating and the total number of ratings for all books published by Friedman/Fairfax Publishing in the year 1961.	**#DSL** query = { "size": 0, "query": { "bool": { "must": [{"match": {"Publisher.keyword": "Friedman/Fairfax Publishing"}}, {"term": {"Year-Of-Publication": 1961}}] } }, "aggs": { "average_rating": {"avg": {"field": "Book-Rating"}}, "total_ratings": {"value_count": {"field": "Book-Rating"}} } } **#Post-processing Code** [...]	**#DSL** query = { "size": 0, "query": { "bool": { "must": [{"match": {"Publisher.keyword": "Friedman/Fairfax Publishing"}}, {"term": {"Year-Of-Publication": 1961}}] } }, "aggs": { "average_rating": {"avg": {"field": "Book-Rating"}}, "sum_rating": "sum": "field": "Book-Rating" } } **#Post-processing Code** [...] ##"sum" must be "value_count".
IJE	Calculate the average number of oxygen atoms in single-bonded molecules.	**#DSL1** query_toxicology_bond={ "query": { "term":{ "bond_type.keyword": "-" } } } **#Post-processing code** [...] **#DSL2** query_toxicology_atom={ "query": { "term": { "element.keyword": "o" } } } **#Post-processing code** [...] **#index-join** joined_data=pd.merge(df_bond, df_atom, on="molecule_id",how="inner") **# Post-processing code** [...]	**#DSL1** query_toxicology_bond={ "query": { "term": { "bond_type.keyword": "-" } } } **#Post-processing code** [...] **#DSL2** query_toxicology_atom={ "query": { "term": { "element.keyword": "o" } } } **#Post-processing code** [...] **# index-join** {NULL} **# Post-processing code** [...] ##comment: No index-join was performed.
GWPC	Calculate the average score for each post category, and list the categories with an average score not less than 47 along with their corresponding average scores.	**# DSL** query={...} **#Post-processing code** index_name= "movies_posts_comments" response=esclient.search (index=index_name, body=query) result=[(bucket['key'], bucket['avg_score']['value']) for bucket in response ['aggregations'] ['tags']['buckets']]	**# DSL** query={...} **#Post-processing code** index_name= "movies_posts_comments" response=esclient.search (index=index_name, body=query) result=[(bucket['key'],) for bucket in response ['aggregations'] ['tags']['buckets']] ##comment: should be (bucket['key'], bucket['avg_score']['value'])

Table A4: Examples of three main error types. correct, incorrect, and ## comment is colored.

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 A5. 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 query 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 A5: Features Comparison between Elasticsearch and MySQL

E.2 Efficiency Analysis: ES vs SQL

Our experimental setup includes equivalent query Q_sql and Q_dsl for SQL and DSL. We measured the average execution time of the query 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 A11, 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.

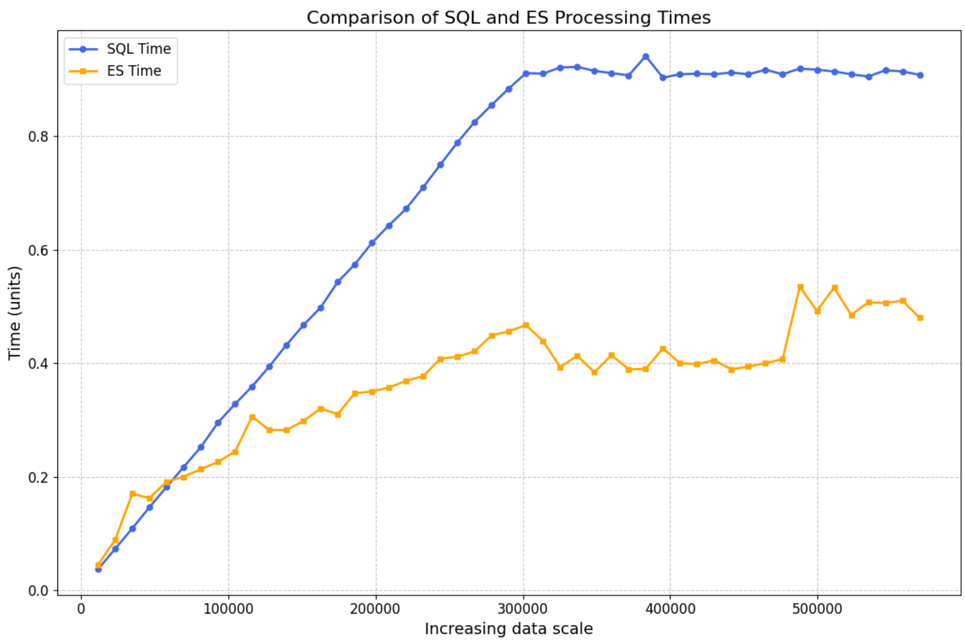

Figure A11: Query time trend chart for SQL and ES with equivalent query statements as data scale grows

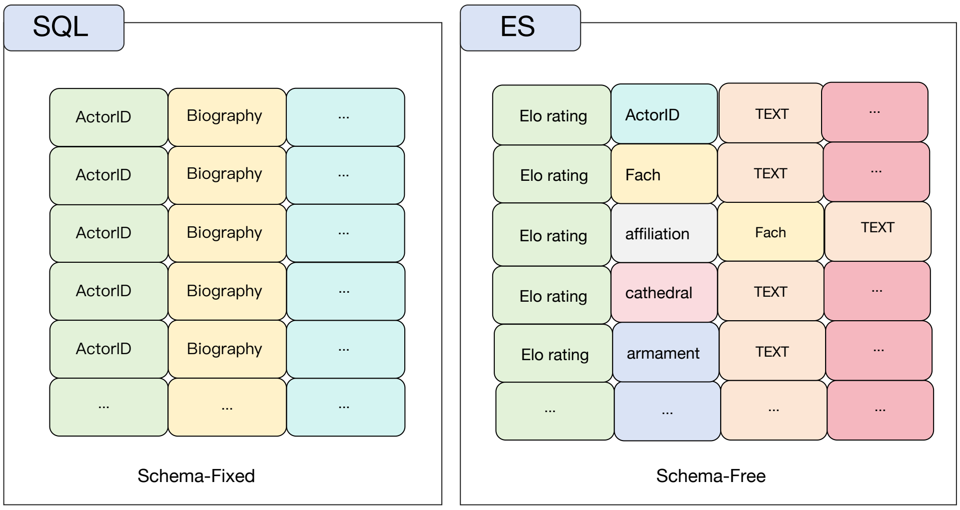

Figure A12: Schema-Free and Schema-Fixed

E.3 Detail for Schema-Free

ES features a schema-free index structure that allows for highly flexible data storage, enabling completely different structures for any two pieces of data within the same index. In contrast, SQL uses a schema-fixed table structure, which only permits data storage according to the initially defined schema. Figure A12 shows the SQL table structure on the left, where each row has a uniform schema. On the right is the ES index structure, where each row can have a different schema; for example, the first row includes Elo rating, ActorID, and TEXT, while the second row uses a different schema with ActorID replaced by Fach.