DualES: Internalizing Query-Text Duality for Text-to-ES via Bidirectional Reasoning-Enhanced Supervision

Anonymous ACL submission

Abstract

Text-to-ES aims to translate natural language into executable Elasticsearch(ES) queries. Unlike more open-ended reasoning tasks, such as mathematical reasoning, this problem is defined over a constrained domain-specific language, where the synergy between forward reasoning and backward reasoning can be more explicitly realized under strict structural constraints. While recent work has explored reasoning-oriented training and prompting in text-to-SQL, forward and backward reasoning have not been thoroughly studied in text-to-ES. In this paper, we systematically study the **bidirectional duality** of the text-to-ES task by establishing a bidirectional reasoning perspective across both training and prompting stages. We introduce two large-scale datasets, **TERD** and **ETRD**, for forward and backward reasoning, respectively, and propose **DualES**, a unified framework that combines the two. Experimental results show that our DualES-CoRE strategy sets a new State-of-the-Art(**SOTA**), achieving 69.23% on LED and 50.14% on BirdES. This corresponds to absolute execution accuracy gains of 6.92% and 7.34% over previous benchmarks, outperforming both open-source and closed-source baselines. We will release the datasets in the future.

1 Introduction

Text-to-Query (Xue et al., 2025) translation serves as a pivotal bridge for seamless interaction between users and databases. Recently, Large Language Models (LLMs) have significantly advanced this field (Li et al., 2023). In text-to-SQL, to better capture user intent and assist query generation, current research has primarily focused on enhancing performance through schema selection (Lee et al., 2025b; Wang et al., 2025c; Hao et al., 2025) and instruction tuning (He et al., 2025a; Dai et al., 2025b). These methods are also applicable to text-to-ES.

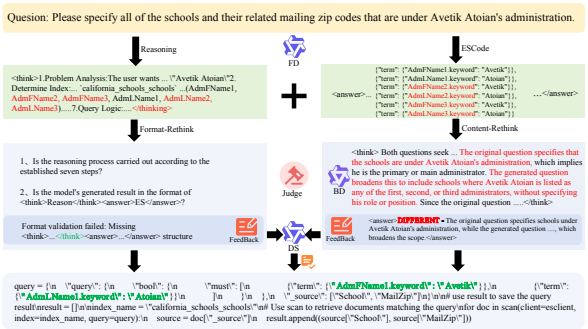

Figure 1: "Fine-tuned on TERD and ETRD, the Qwen-based models FD and BD generate ES queries and reconstruct natural language explanations, respectively. Subsequently, a Judge module evaluates both the reasoning process and the final output through Format-Rethink and Content-Rethink mechanisms.

However, these methods predominantly treat text-to-Query as a unidirectional reasoning process, which often fails to ensure that the generated query remains strictly consistent with the user's original intent. As illustrated in Figure 1, a model might generate an ES query containing fields such as "AdmFName2" and "AdmLName2"; while syntactically correct, such outputs clearly deviate from the semantic requirements of the user. To mitigate this, exploratory works (Chen et al., 2025a; Peng et al., 2026; Li et al., 2026) have introduced sql-to-Text processes, guiding models to compare generated text with the initial intent. Nonetheless, these approaches remain confined to the prompt engineering level and lack a deeper internalization of the underlying logic.

Drawing a parallel to related work in sql-to-Text, we hypothesize that text-to-Query tasks, such as text-to-ES, inherently exhibit a similar bidirectional duality. The concept of duality was originally pioneered in dual learning (He et al., 2016), where it has been extensively studied within the context of machine translation (Sennrich et al., 2016; Xia et al., 2018). Building on this paradigm, Chen et al.

(2025b) proposed enhancing model performance through backward reasoning.

Inspired by those prior works, we systematically explore the **bidirectional duality** in the Text-to-ES task, characterized as a symmetric semantic mapping between text (T) and structured DSL (E). Unlike divergent mathematical proofs (Zhang et al., 2025) with infinite derivation paths, the structural determinism of Elasticsearch allows the $T \leftrightarrow E$ relationship to be modeled as a rigorous reasoning trace. To this end, we construct a bidirectional pipeline to produce two complementary datasets: **TERD** (68k pairs) and **ETRD** (97k pairs). Building on these, we propose **DualES**, the first framework to introduce dual learning to Text-to-ES via Supervised Fine-Tuning. By training across four progressive strategies—Forward Reasoning (FD), Backward Reasoning (BD), Multi-Task Fine-tuning (MT), and Collaborative Reasoning (CoRE)—DualES internalizes this intrinsic duality directly into the model's parametric knowledge. Extensive experiments on thirty model configurations demonstrate that DualES-CoRE achieves **SOTA** performance. Our model reaches 69.23% execution accuracy on LED and 50.14% on BirdES, surpassing previous SOTA baselines by absolute margins of 6.92% and 7.34%, respectively.Our contributions are:

- To the best of our knowledge, we are the first to explore the **bidirectional duality** in Text-to-ES, establishing a bidirectional reasoning perspective between natural language and ES query.
- We propose a systematic pipeline for constructing high-quality, paired forward and backward reasoning datasets. This process yields **TERD** and **ETRD**, both of which feature explicit reasoning traces.
- We propose **DualES**, a novel framework designed to comprehensively enhance model capabilities in text-to-ES tasks from both diverse training perspectives and advanced prompting strategies.
- Our approach sets a new **SOTA** by achieving 69.23% execution accuracy on LED and 50.14% on BirdES. These results, obtained from broad experiments on thirty model variants, outperform previous bests by 6.92% and 7.34% in absolute terms.

2 Related Work

2.1 LLM Reasoning

Recent advancements in large language models (LLMs) (Kaddour et al., 2023; Lightman et al., 2023; Huang et al., 2024), notably Chain-of-Thought (CoT) and its variants (Kojima et al., 2023; Wang et al., 2023; Yao et al., 2023), have catalyzed research into model reasoning, culminating in high-performing models like GPT-o1, DeepSeek-R1, and Gemini-Pro. Current reasoning research in the code domain typically falls into three categories: (1) Plan-based reasoning, which guides code generation through intermediate natural language plans or problem observations (e.g., PlanSearch (Wang et al., 2024), Self-Planning (Jiang et al., 2023), and ClarifyGPT (Mu et al., 2024)); (2) Structure-based reasoning, which utilizes programming constructs like loops and control flow graphs as reasoning steps, such as SCoT (Wang et al., 2025b), CodeChain (Le et al., 2023), and VisualCoder (Chi et al., 2025); and (3) Reasoning-based fine-tuning, which constructs high-quality CoT data for instruction tuning or optimization, including UniCoder's intermediate representations (Sun et al., 2024), ChainCoder's AST-guided strategy (Zheng et al., 2023), and ExCoT-DPO's iterative optimization (Zhai et al., 2025). Distinct from these approaches, which focus on plan/structure-to-code or instruction fine-tuning, we are the first to investigate the impact of code-to-text reasoning data on text-to-code tasks.

2.2 Knowledge Distillation

Data distillation is pivotal across research domains, evolving from early neural network knowledge distillation (Hinton et al., 2015) to modern techniques leveraging powerful LLMs (e.g., GPT-o1, Gemini, DeepSeek-R1) for high-quality data synthesis (Peng et al., 2023; Li et al., 2024b). In specialized fields, researchers have distilled large-scale reasoning datasets for medicine (Sun et al., 2025; Huang et al., 2026) and mathematics (e.g., TR-CoT). The code domain has similarly benefited from distillation-driven improvements, including the synthesis of positive/negative samples (Dai et al., 2025a), text-to-SQL data augmentation (Yang et al., 2024; Li et al., 2024a), and the construction of massive competition-level reasoning trajectories like rStar-Coder (Liu et al., 2025b) and instruction-code quadruples (Abed et al., 2025). Furthermore, distillation has been applied to spe-

cific tasks like text-to-ES template generation (Xue et al., 2025). Distinct from these efforts, our work is the first to distill reasoning datasets specifically for the text-to-ES domain and introduces the inaugural code-to-text reasoning dataset for Elasticsearch.

3 Problem Formulations

We define two complementary bidirectional tasks that incorporate reasoning chains for enhanced interpretability: Text-to-ReasonES and ES-to-ReasonText.

In **Text-to-ReasonES**, given a natural language text T and schema information $\mathcal{S}$, the model generates a reasoning chain $\mathcal{R}$ followed by an ES query $\mathbf{E}$:

$$\mathcal{R}, \mathbf{E} = f(T, \mathcal{S} \mid \boldsymbol{\theta}). \tag{1}$$

Conversely, **ES-to-ReasonText** converts an ES query $\mathbf{E}$ back into a reasoning chain $\mathcal{R}$ and a corresponding natural language description T':

$$\mathcal{R}, T' = g(\mathbf{E}, \mathcal{S} \mid \boldsymbol{\phi}). \tag{2}$$

The objective is to internalize the duality between these tasks, ensuring semantic consistency in the bidirectional transformation $T \leftrightarrow \mathbf{E}$.

4 Methodology

4.1 Overview

As illustrated in Figure 2 , our framework orchestrates a comprehensive pipeline for bidirectional reasoning distillation, encompassing five core phases: (1) Reasoning Process Distillation, where intermediate trajectories are extracted from multiple teacher models; (2) Reasoning Data Verification, which ensures the logical integrity of the distilled content; (3) Multi-dimensional Data Annotation, for categorizing query types and difficulty; (4) Quality Evaluation, involving rigorous scoring across predefined metrics; and (5) DualES Framework, to optimize the student model's performance. The primary objective is to distill high-fidelity reasoning paths from diverse frontier LLMs, ultimately empowering the student model with robust, bidirectional reasoning capabilities. The corresponding prompts are shown in Appendix E.

4.2 Reasoning Process Distillation

Inspired by the STaR (He et al., 2025b), which leverages iterative correctness feedback to refine model reasoning, we adapt its core principles to curate a reasoning-centric corpus. The first dataset, TERD (Text-to-ES Reasoning Dataset), focuses on distilling intermediate reasoning trajectories from a suite of advanced LLMs, including Qwen3-max-preview, DeepSeek-V3.2, ChatGLM-4.7, and GPT-4.1-Nano. To enhance data diversity, we employ two complementary construction strategies: one utilizing explicitly defined pipelines and another based on automatically induced workflows. Through iterative distillation, we curated 68,044 high-quality $\langle T, R, E \rangle$ triples. The second dataset, ETRD (From ES to Text Reasoning Dataset), reverses the reasoning direction by distilling $\langle E, R, T \rangle$ triples using the same LLM suite. We implement an iterative dual-validation mechanism to rigorously verify both the synthesized reasoning trajectories and the final natural language descriptions, yielding 97,290 high-quality instances. The two distillation processes can be formally defined as follows:

$$\begin{aligned} \mathcal{D}_{TERD} &= \{(t, r, e) \mid r, e \leftarrow \phi_{LLM}(t)\} \\ \mathcal{D}_{ETRD} &= \{(e, r, t) \mid r, t \leftarrow \psi_{LLM}(e)\} \end{aligned} \tag{3}$$

where ϕ and ψ represent the forward (Text-to-ES) and backward (ES-to-Text) distillation mappings, respectively, and $t \in T, r \in R, e \in E$.

4.3 Verification and Iteration

To ensure the technical accuracy of the TERD dataset, we employ a Python-based interpreter to verify if the generated ES code executes correctly against the database. Beyond execution, we utilize Qwen3-32B as a specialized critic to audit the reasoning process, ensuring it adheres to our seven-pillar validation criteria, detailed in Appendix A.1. For the ETRD dataset, Qwen3-32B serves as the central judge to evaluate the semantic alignment between the generated natural language and the source ES query. The verification follows a structured seven-step detailed in Appendix A.2. Empirical observations indicate that TERD data volume converges around the sixth iteration, whereas ETRD converges more rapidly, stabilizing by the fourth iteration. The iterative verification process for a dataset $\mathcal{D}$ at iteration k can be formally defined as:

$$\mathcal{D}^{(k)} = \{(x, r, y) \mid \mathcal{V}_{exe}(y) \wedge \mathcal{V}_{crit}(r, x, y)\}_{k=1}^{K} \tag{4}$$

where $\mathcal{V}_{exe}$ denotes the execution-based interpreter and $\mathcal{V}_{crit}$ represents the multi-dimensional LLM-based critic. The convergence is reached when

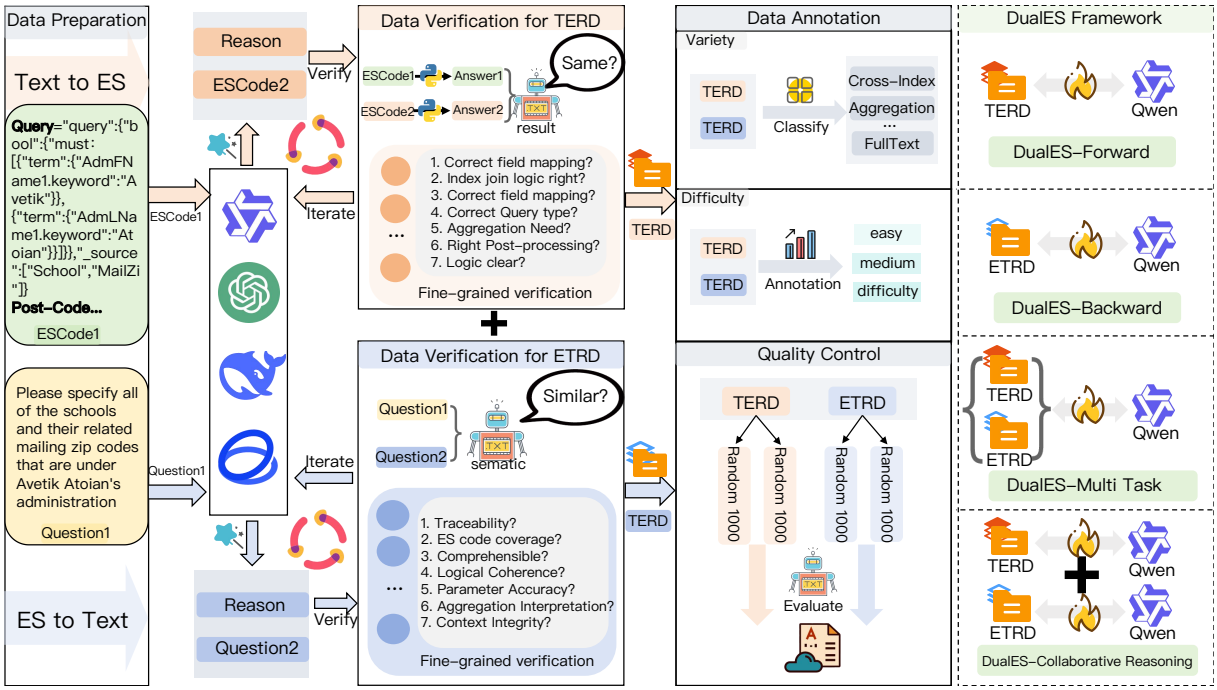

Figure 2: Detailed flowchart of data construction and DualES framework.

$|\mathcal{D}^{(k)}| \approx |\mathcal{D}^{(k-1)}|$, specifically at $K_{terd} = 6$ and $K_{etrd} = 4$.

4.4 Data Annotation

To facilitate fine-grained data management and high-quality filtering, we perform comprehensive annotation across two dimensions: Query Taxonomy and Complexity Grading. Following ES official documentation, we categorize queries into six functional types: Specialized, Term-Level, Full-Text, Geography, Joining, and Aggregation (Xue et al., 2025). We further differentiate between single-index queries and multi-index queries based on the data scope. Additionally, leveraging Qwen3-32B, we classify each instance into three difficulty levels—easy, medium, and difficult—based on the number of query conditions and the structural complexity of the code. The detailed data classification and difficulty distribution are presented in Appendix A.3. Formally, for any instance $d_i \in \mathcal{D}_{TERD} \cup \mathcal{D}_{ETRD}$, the annotation process $\mathcal{A}$ is defined as:

$$\mathcal{A}(d_i) = \langle \text{Type}(S), \text{Scope}(S), \text{Diff}(S, R) \rangle \quad (5)$$

where Type $\in \{S, T, F, G, J, A\}$ (Specialized, Term-level, Full-text, Geography, Joining, Aggregation), Scope $\in \{S, M\}$ (Single-index, Multi-index), and Diff $\in \{E, M, D\}$ (Easy, Medium, Difficult) represent the query category, data scope, and structural complexity respectively.

4.5 Quality Evaluation

To quantitatively assess the bidirectional reasoning datasets, we implemented an automated quality evaluation module. Using Qwen3-32B as the evaluator, each sample is scored across eight predefined dimensions. Given the substantial scale of the dataset, we adopted a stratified random sampling approach, selecting two independent sets of 1,000 samples from both TERD and ETRD. The final quality metric for each dataset is defined as the mean score across these two evaluation rounds, as presented in Table 1. This rigorous evaluation process ensures the reliability of the distilled reasoning trajectories before they are used for model training. Formally, let $\mathcal{Q}(d_i) \in [0, 1]^8$ be the eight-dimensional score vector for an instance d_i. For each dataset $\mathcal{D} \in \{\mathcal{D}_{TERD}, \mathcal{D}_{ETRD}\}$, the final quality score $\bar{Q}_{\mathcal{D}}$ is calculated as:

$$\bar{Q}_{\mathcal{D}} = \frac{1}{2} \sum_{j=1}^{2} \left(\frac{1}{|S_j|} \sum_{d_i \in S_j} \text{avg}(\mathcal{Q}(d_i)) \right) \quad (6)$$

where $S_j \subset \mathcal{D}$ denotes the j-th independent stratified sample set ($|S_j| = 1,000$), and $\text{avg}(\cdot)$ represents the arithmetic mean across the eight evaluation dimensions.

4.6 DualES Framework

To comprehensively investigate the Text-to-ES bidirectional duality from both training and prompt-

Dataset	Samples	Avg. Tokens	Scoring
TERD	68,044	695.75	6.19
ETRD	97,290	540.54	6.23

Table 1: Statistics of TERD, ETRD Datasets

ing perspectives, we propose the **DualES** framework. This framework encompasses four progressive strategies: three training-based paradigms (**DualES-FD**, **DualES-BD**, and **DualES-MT**) and one prompting-based mechanism (**DualES-CoRE**). As illustrated in Figure 2, the former three focus on internalizing bidirectional dual knowledge through supervised fine-tuning, while the latter leverages collaborative reasoning to further enhance the model's inference capabilities.

(1) **DualES-FD** (Forward Reasoning): A baseline strategy focused exclusively on the Text-to-ReasonES task using the TERD dataset to optimize the forward mapping $T \rightarrow \langle \mathcal{R}, \mathbf{E} \rangle$.

(2) **DualES-BD** (Backward Reasoning):A baseline strategy focused exclusively on the ES-to-ReasonText task using the ETRD dataset to optimize the backward mapping $\mathbf{E} \rightarrow \langle \mathcal{R}, T \rangle$.

(3) **DualES-MT** (Multi-Task Fine-tuning): A unified strategy where a single model is fine-tuned on both TERD and ETRD datasets, encouraging the internalization of bidirectional semantic mappings through shared parameters.

(4) **DualES-CoRE** (Bidirectional Collaborative Reasoning): Our proposed refinement framework. Unlike the one-pass strategies, DualES-CoRE utilizes Model B as a semantic validator to provide feedback for Model A. Formally, the reasoning state at iteration t is updated as:

$$(\mathcal{R}, \mathbf{E})^{(t+1)} = \text{Model}_A \left(T, \mathcal{S}, \text{FB}(\mathbf{E}^{(t)}, T) \right) \quad (7)$$

where $\text{FB}(\mathbf{E}, T)$ is the feedback function defined by the conjunction of structural and semantic consistency:

$$\text{FB}(\mathbf{E}, T) = \mathcal{V}_{\text{str}}(\mathbf{E}) \wedge \mathcal{V}_{\text{sem}}(\text{Model}_B(\mathbf{E}), T) \quad (8)$$

Here, $\mathcal{V}_{\text{str}}$ and $\mathcal{V}_{\text{sem}}$ represent the structural validator and the semantic consistency judge performed by Model B, respectively.

5 Experiment Setting

Train Data and Train Models Finally, we constructed the TERD and ETRD inference datasets (detailed statistics in Table 1), on which we trained our models **Qwen2.5-Coder-14B-FD, -BD, -MT, and -CoRE**, based on the four DualES strategies applied to Qwen2.5-Coder-14B-Instruct. The models are fine-tuned for 3 epochs using LoRA. All experiments are conducted on eight NVIDIA A100 (80GB) Tensor Core GPUs. Further details in Appendix C.1.

Baseline Models and Methods We compare our models with 1) **General LLMs**, including the Qwen2.5 series (Qwen et al., 2025) (14B, 32B, 72B-Instruct), GLM-4-9B-0414 (Glm et al., 2024), and the Qwen3 series (Yang et al., 2025) (4B, 8B, 14B, 32B, and 30B-A3B); 2) **Code LLMs**, comprising Qwen2.5-Coder (Hui et al., 2024) (14B, 32B-Instruct) and Qwen3-Coder-30B-A3B-Instruct; 3) **Advanced Models**, including the GPT series (Achiam et al., 2023) (4o-mini, 5-mini, 4.1-Nano), Claude 4.6 Sonnet, Gemini-2.5-pro, DeepSeek series (Guo et al., 2025) (V3.1, V3.2, Reasoner), Qwen3-max-preview, Qwen3.5-plus, and MiniMax-M2.5; 4) **Fine-tuned Baselines**, Qwen2.5-Coder-14B-FeynMan (Xue et al., 2025) and Qwen3-8B, 14B, 32B-SFT; and 5) **Prompting Methods**, such as DIN-SQL (Pourreza and Rafiei, 2023), MAC-SQL (Wang et al., 2025a), and MCS-SQL (Lee et al., 2025a). All experiments in the main results are conducted using a one-shot approach. For ICL selection, we utilize the Qwen3-Embedding-0.6B model to select one example from the train set. Further inference details in Appendix C.2.

5.1 Datasets and Evaluations

As illustrated in Table 2, we evaluate LLMs on two datasets: **LED** and **BirdES** Xue et al. (2025). To assess the generated queries, we employ three key metrics: Domain-Specific Language Exact Match (**DSLEM**) for literal string identity; Execution Accuracy (**EX**) for functional correctness via result set comparison; and Valid Efficiency Score (**VES**), which balances execution accuracy with performance profiling to measure computational efficiency relative to the ground truth.

5.2 Main Results

As shown in Table 2, we evaluate our models across two benchmarks: Overall: **1) Comparison with Advanced Models.** Our DualES-CoRE strategy achieves SOTA performance with 69.23 EX on LED and 50.14 EX on BirdES, surpassing all pro-

Models	LED			BirdES		
	DSL-EM	EX	VES	DSL-EM	EX	VES
General Models						
Qwen2.5-14B-Instruct†	2.14	15.91	16.25	1.62	16.96	19.68
Qwen2.5-32B-Instruct†	8.12	27.55	29.11	2.05	23.34	26.74
Qwen2.5-72B-Instruct†	27.95	43.28	43.86	2.93	25.03	26.74
GLM-4-9B-0414	25.67	44.90	44.91	1.17	11.31	12.73
Qwen3-4B	28.20	41.33	42.17	2.20	11.31	11.87
Qwen3-8B	27.42	44.18	44.91	1.40	13.58	13.86
Qwen3-14B	27.81	45.33	47.28	1.47	14.02	14.31
Qwen3-30B-A3B	26.84	47.87	46.44	4.19	22.69	22.97
Qwen3-32B	30.21	48.21	48.37	3.67	31.20	32.10
Code Models						
Qwen2.5-Coder-14B-Instruct†	2.22	19.49	21.64	3.30	22.32	22.96
Qwen2.5-Coder-32B-Instruct†	26.57	43.79	44.84	3.34	24.81	28.16
Qwen3-Coder-30B-A3B-Instruct	26.90	50.21	51.41	3.16	32.16	33.72
Advanced Models						
MiniMax-M2.5	22.08	44.19	44.20	3.01	25.55	25.65
GPT-5-mini	25.55	44.55	43.64	3.04	40.41	42.33
GPT-4o-mini	21.45	43.87	44.09	5.13	27.01	27.24
GPT-4o†	30.15	48.73	49.42	7.78	42.80	44.89
GPT-4.1-Nano	27.42	60.86	62.17	3.01	23.26	23.48
Qwen3-max-preview	29.04	61.02	61.39	8.34	48.97	50.27
Qwen3.5-plus	29.62	61.73	60.08	6.69	45.22	45.48
Gemini-2.5-pro	24.36	58.65	30.35	5.65	49.77	**51.03**
Claude4.6 Sonnet	33.04	60.86	61.96	8.07	49.12	47.35
Deepseek-V3.1	25.53	55.04	56.35	5.36	36.12	36.71
Deepseek-V3.2	26.97	59.84	60.58	4.04	41.77	42.27
Deepseek-Reasoner	22.15	61.71	62.60	5.07	46.39	46.94
Fine-tuning Models						
Qwen2.5-Coder-14B-FeynMan†	**48.27**	62.31	63.25	7.05	40.61	37.82
Qwen3-8B-SFT	35.43	50.81	51.97	2.87	24.8	26.8
Qwen3-14B-SFT	41.80	60.70	61.13	7.92	40.24	41.87
Qwen3-32B-SFT	36.73	63.81	62.84	8.72	44.99	46.22
Qwen2.5-Coder-14B-MT(ours)	26.82	67.61	68.35	4.63	48.50	50.20
Prompting Methods						
MAC+GPT-4o mini	3.44	36.39	38.04	3.40	9.03	13.91
MCS+Qwen3-8B	21.77	45.61	45.88	1.39	9.32	13.01
DIN+Qwen3-8B	0.39	10.20	10.84	0.00	3.52	4.78
DualES-CoRE(ours)	27.31	**69.23**	**68.96**	**8.81**	**50.14**	50.86

Table 2: Performance comparison on LED and BirdES benchmarks. † denotes the result from Xue et al. (2025). **Bold** denotes the best performance. Underline denotes the second performance.

prietary models, including the strongest baselines Qwen3.5-plus (61.73) and Gemini-2.5-pro (49.77). **2) Comparison with Fine-tuning Models.** Compared to the FeynMan baseline, which is limited by suboptimal schema selection, DualES-CoRE achieves significant performance boosts of +6.92% and +9.53% in terms of EX on LED and BirdES, respectively. Similarly, Qwen2.5-Coder-14B-MT demonstrates robust improvements, yielding gains of +5.3% and +7.89% EX on the same benchmarks. This demonstrates the effectiveness of our DualES framework, showcasing improvements in both training methodology and prompting strategies. **3) Comparison with Qwen Series.** We observe a clear generational leap: Qwen3-32B comprehensively outperforms the larger Qwen2.5-72B-Instruct by 3.37% on LED and 7.13% on BirdES. While Qwen3-Coder-30B-A3B is the top open-source baseline, it still lags significantly behind our Qwen2.5-Coder-14B-MT model and DualES-CoRE strategy. **4) Comparison with Prompting Methods.** Standard Text-to-SQL prompting methods (e.g., DIN, MAC) generally underperform in the ES domain. *MCS* is the sole exception, yielding a 1.43% improvement on LED likely due to its test-time-scaling (Liu et al., 2025a) nature. This results suggest that Text-to-ES demands deep structural adaptation rather than simple prompt engineering.

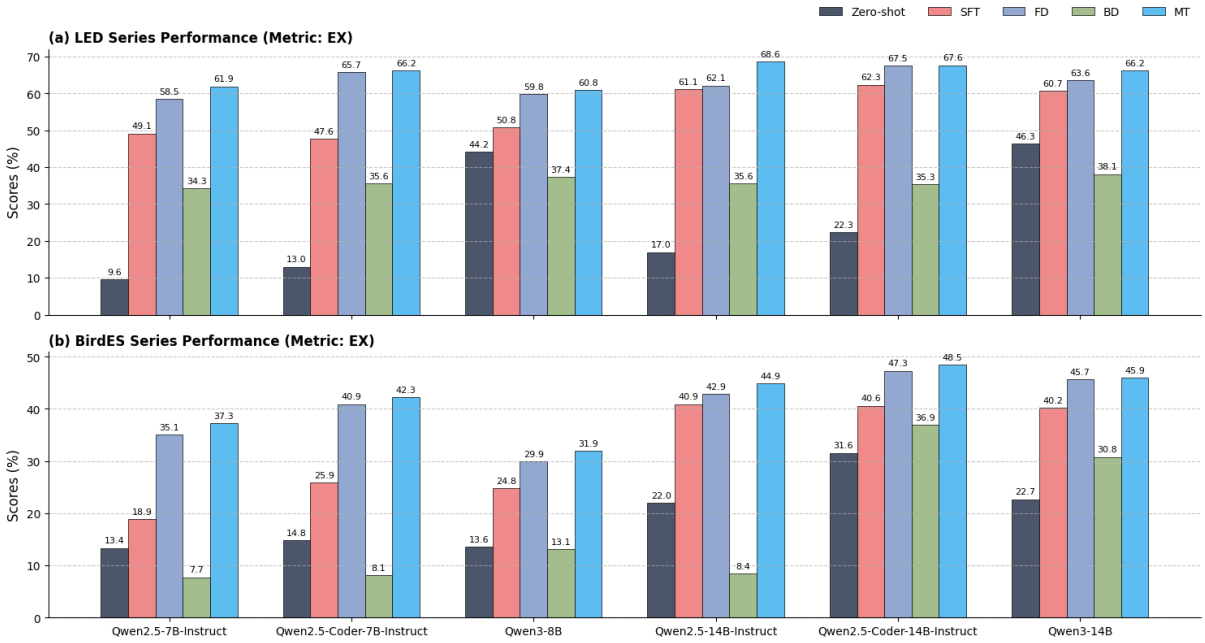

Figure 3: Performance of various models on LED and BirdES benchmarks under different DualES training strategies.

5.3 Ablation Study

To explore the bidirectional duality within the Text-to-ES task, we conducted experiments using the Qwen2.5 (7B, 14B, Coder-7B, Coder-14B) and Qwen3 (8B, 14B) series under various training configurations. These include Zero-shot, Fine-tuning (Direct SFT), DualES-FD, Dual-BD, and Dual-MT. The results, as illustrated in Fig. 3, lead to the following observations: **1) SFT significantly enhances model performance.** Compared to the zero-shot baseline, direct fine-tuning yields a substantial leap across all models. For instance, the EX score of Qwen2.5-14B on the LED task surged from 16.96% to 61.10%, demonstrating the necessity of task-specific adaptation for complex mapping tasks. **2) Dual-FD further breaks the performance ceiling.** Building upon the Direct SFT, the FD variants achieved superior results across the entire Qwen series. Specifically, Qwen2.5-Coder-7B witnessed an EX score increase from 47.63% to 65.72% on the LED task, which validates the effectiveness of our synthesized TERD dataset in strengthening logical reasoning. **3) Dual-BD further boosts model performance when the model possesses strong coding capabilities.** Overall, incorporating backward reasoning in isolation—without simultaneous forward training—tends to decay downstream performance. For example, the top-performing Qwen3-14B saw an EX score drop of approximately 8.27% on the LED dataset compared to direct inference. Intriguingly, Qwen2.5-Coder-14B exhibited positive gains on both datasets under Dual-BD. This suggests that the latent mapping information within the ES-to-Text data can provide positive gradients for Text-to-ES tasks only when the model has sufficient parameter capacity or domain-specific prior knowledge (e.g., code pre-training) to decode such inverse relationships. **4) Dual-MT offers differential advantages based on model capacity.** Our results indicate that simultaneous learning of both directions typically surpasses single-direction logic. Notably, for stronger models like the Qwen3 series and Coder-14B, the performance increment from FD to MT is relatively marginal. In contrast, weaker models (e.g., Qwen2.5-7B and 14B) exhibit more pronounced improvements. We hypothesize that while stronger models may reach a knowledge saturation point through the TERD dataset alone, smaller models rely more heavily on the ETRD path for essential knowledge compensation.

5.4 Analysis on Dual-CoRE

To investigate the impact of different modules within CoRE on task performance, we further conduct ablation studies on the Format-Rethink, Content-Rethink and Self-Rethink modules. We use Qwen2.5-Coder-7B-FD and Qwen2.5-Coder-14B-MT as the forward inference models. The experimental results are summarized in Table 3. We observed that:**1) The gains from Format-Rethink**

Models	LED			BirdES		
	DSLEM	EX	VES	DSLEM	EX	VES
Qwen3-14B-FD	34.60	63.60	65.45	5.15	45.70	46.40
Qwen2.5-14B-FD	26.60	62.12	63.18	6.14	42.90	43.10
Qwen2.5-Coder-14B-FD	31.72	67.51	68.21	7.31	47.30	47.60
Qwen2.5-Coder-14B-MT	26.82	67.61	68.35	4.63	48.50	50.20
Qwen2.5-Coder-14B-CoRE♠ (Format-Rethink)	32.13	67.89	68.64	7.23	47.54	47.33
Qwen2.5-Coder-14B-CoRE♣ (Format-Rethink)	29.32	68.26	68.65	5.12	48.73	49.17
Qwen2.5-Coder-14B-CoRE♠ (Content-Rethink)	29.29	68.94	68.87	7.34	48.15	49.26
Qwen2.5-Coder-14B-CoRE♣ (Content-Rethink)	27.31	69.23	68.96	8.81	50.14	50.86
Qwen2.5-Coder-14B-FD (Self-Rethink)	29.10	66.80	68.40	3.52	44.00	44.20
Qwen2.5-14B-FD (Self-Rethink)	25.90	55.70	57.40	3.01	40.30	42.00
Qwen3-14B-FD (Self-Rethink)	17.50	38.00	38.90	4.62	45.70	46.60

Table 3: ♠ and ♣ denote base models are Qwen2.5-Coder-14B-FD and Qwen2.5-Coder-14B-MT, respectively.

were minimal. This suggests that the model has already internalized the predefined reasoning paths during the supervised fine-tuning (SFT) process, making additional format-level self-reflection largely redundant. **2) Content-Rethink yielded a substantial performance boost.** This observation aligns with the findings of Chen et al. (2025a), Peng et al. (2026) and Li et al. (2026) in the Text-to-SQL domain. By prompting the model to translate the generated ES query back into the original natural language text, the system can perform verification at the semantic level. This process effectively identifies and rectifies logical discrepancies between the user's intent and the query structure. Additionally, we evaluate and compare the effectiveness of CoRE against the Self-Rethink approach.**3) Self-Rethink leads to a performance degradation.** Instead of improving accuracy, the model's performance decreased after applying the Self-Rethink strategy. This aligns with recent studies (Huang et al., 2023) which demonstrate that large language models often lack the intrinsic capability for effective self-correction in the absence of external feedback signals. The complete case are shown in Table 4.

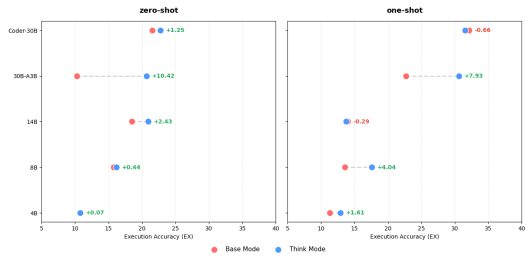

Figure 4: The impact of think mechanisms on model performance under zero-shot and one-shot settings.

5.4.1 Analysis on Think Mechanisms

To evaluate the impact of the *think* mechanism on Text-to-ES tasks, we compared five models ranging from Qwen-4B to Qwen3-Coder-30B-A3B (see Figure 4). Our analysis reveals that the **think mode generally enhances model performance,** with a particularly pronounced boost observed in Qwen-30B-A3B. This suggests that the reasoning process effectively unlocks the model's latent potential, enabling Qwen-30B-A3B to achieve performance levels nearly on par with Qwen3-Coder-30B.

5.5 Analysis on Data Synthesis Strategies

We evaluate models fine-tuned on LLM-Generation versus Human-Assignment strategies using the LED benchmark (Figure 5). Results indicate that **both datasets yield comparable performance**; across zero-shot (blue) and one-shot (red) settings, the gap between LLM-generated data (light shades) and human assignments (dark shades) remains marginal. Specifically, the two approaches achieve competitive EX scores of 66.85% and 67.50%, respectively. This demonstrates that synthesized prompts can effectively match the quality of manual assignments for task-specific fine-tuning.

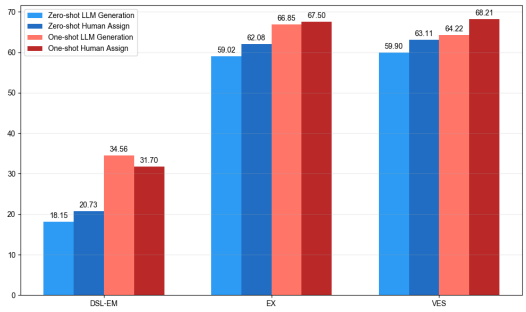

Figure 5: Performance comparison on LED benchmark between LLM-Generation and Human-Assignment.

6 Conclusion

This paper comprehensively explores the bidirectional duality of the Text-to-ES task through both training and prompting perspectives. We introduce a systematic pipeline to construct two large-scale reasoning datasets, TERD and ETRD. Based on these, we propose the DualES framework, which integrates three training strategies (FD, BD, MT) and an prompting strategy (CoRE). Extensive experiments validate that leveraging such duality significantly boosts performance, demonstrating the effectiveness of the DualES framework in internalizing and applying bidirectional dual knowledge.

Limitations

Despite the significant performance gains achieved by DualES, this study has several limitations: (1) Language and Domain Coverage: While TERD and ETRD cover a wide range of search scenarios, they are primarily focused on English and general-purpose datasets; their zero-shot transferability to highly specialized vertical domains, such as medical or legal indexing, requires further investigation. (2) Inference Overhead: Although offline SFT internalizes duality into model weights, the dual-model collaborative inference mechanism introduces additional computational costs compared to a single forward model, which may pose challenges for latency-sensitive real-time applications. (3) Dynamic Schema Adaptation: Our framework assumes relatively stable database schemas; for environments with frequently evolving data structures, incorporating more flexible retrieval-augmented generation (RAG) techniques could further enhance the model's adaptability without frequent retraining.

Ethical Considerations

This work focuses on the Text-to-ES task and conducts experiments on two open-sourced datasets (LED and BirdES), which have been published and provided for scientific research. Our TERD and ETRD do not involve any privacy concerns. We will release these datasets publicly for research purposes in the future. To our knowledge, there are no other potential ethical implications associated with the proposed datasets.

References

Amal Abed, Ivan Lukic, Jörg K. H. Franke, and Frank Hutter. 2025. Increasing llm coding capabilities through diverse synthetic coding tasks. *Preprint*, arXiv:2510.23208.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Daojun Chen, Xi Wang, Shenyuan Ren, Qingzhi Ma, Pengpeng Zhao, and An Liu. 2025a. Gbv-sql: Guided generation and sql2text back-translation validation for multi-agent text2sql. *arXiv preprint arXiv:2509.12612*.

Justin Chen, Zifeng Wang, Hamid Palangi, Rujun Han, Sayna Ebrahimi, Long Le, Vincent Perot, Swaroop Mishra, Mohit Bansal, Chen-Yu Lee, et al. 2025b. Reverse thinking makes llms stronger reasoners. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8611–8630.

Cuong Le Chi, Chau Truong Vinh Hoang, Phan Nhat Huy, Dung D. Le, Tien N Nguyen, and Nghi D. Q. Bui. 2025. VisualCoder: Guiding large language models in code execution with fine-grained multimodal chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6643–6660, Albuquerque, New Mexico. Association for Computational Linguistics.

Chengwei Dai, Kun Li, Wei Zhou, and Songlin Hu. 2025a. Capture the key in reasoning to enhance CoT distillation generalization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 441–465, Vienna, Austria. Association for Computational Linguistics.

Yaxun Dai, Haiqin Yang, Mou Hao, and Pingfu Chao. 2025b. PARSQL: Enhancing text-to-SQL through SQL parsing and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 661–681, Vienna, Austria. Association for Computational Linguistics.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhifeng Hao, Junqi Huang, Shaobin Shi, Ruichu Cai, and Boyan Xu. 2025. GenLink: Generation-driven schema-linking via multi-model learning for text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 29892–29905, Suzhou, China. Association for Computational Linguistics.

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. *Advances in neural information processing systems*, 29.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025a. Star-sql: Self-taught reasoner for text-to-sql. *Preprint*, arXiv:2502.13550.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025b. STaR-SQL: Self-taught reasoner for text-to-SQL. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*

Papers), pages 24365–24375, Vienna, Austria. Association for Computational Linguistics.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. *Preprint*, arXiv:2310.01798.

Xiaoke Huang, Juncheng Wu, Hui Liu, Xianfeng Tang, and Yuyin Zhou. 2026. m1: Unleash the potential of test-time scaling for medical reasoning with large language models. *Preprint*, arXiv:2504.00869.

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

Xue Jiang, Yihong Dong, Lecheng Wang, Qiwei Shang, and Ge Li. 2023. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33:1 – 30.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *Preprint*, arXiv:2307.10169.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

Hung Le, Hailin Chen, Amrita Saha, Akash Gokul, Doyen Sahoo, and Shafiq R. Joty. 2023. Codechain: Towards modular code generation through chain of self-revisions with representative sub-modules. *ArXiv*, abs/2310.08992.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025a. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353.

Jihyung Lee, Jin-Seop Lee, Jaehoon Lee, YunSeok Choi, and Jee-Hyong Lee. 2025b. DCG-SQL: Enhancing in-context learning for text-to-SQL with deep contextual schema link graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15397–15412, Vienna, Austria. Association for Computational Linguistics.

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. *Preprint*, arXiv:2402.16347.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface. *A big bench for large-scale database grounded text-to-SQLs*, 2305.

Ming Li, Lichang Chen, Jiuhai Chen, Shwai He, Jiuxiang Gu, and Tianyi Zhou. 2024b. Selective reflection-tuning: Student-selected data recycling for LLM instruction-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16189–16211, Bangkok, Thailand. Association for Computational Linguistics.

Xiang Li, Jinguo You, Heng Li, Jun Peng, Xi Chen, Ziheng Guo, Kun Li, and Tianyi Xu. 2026. G^2sql: Guided & guarded text-to-sql generation with two-stage verification. *Expert Systems with Applications*, page 131276.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Yexiang Liu, Zekun Li, Zhi Fang, Nan Xu, Ran He, and Tieniu Tan. 2025a. Rethinking the role of prompting strategies in llm test-time scaling: A perspective of probability theory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27962–27994.

Yifei Liu, Li Lyna Zhang, Yi Zhu, Bingcheng Dong, Xudong Zhou, Ning Shang, Fan Yang, and Mao Yang. 2025b. rstar-coder: Scaling competitive code reasoning with a large-scale verified dataset. *Preprint*, arXiv:2505.21297.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, Chenxue Wang, Shichao Liu, and Qing Wang. 2024. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proc. ACM Softw. Eng.*, 1(FSE):2332–2354.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *Preprint*, arXiv:2304.03277.

Jun Peng, Jinguo You, Xiang Li, Jiaman Ding, and Lianyin Jia. 2026. Kadp-sql: Knowledge-augmented generation and dual-path validation in text-to-sql. *EAI Endorsed Transactions on Scalable Information Systems*, 12(7).

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, 36:36339–36348.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Tao Sun, Linzheng Chai, Jian Yang, Yuwei Yin, Hongcheng Guo, Jiaheng Liu, Bing Wang, Liqun Yang, and Zhoujun Li. 2024. UniCoder: Scaling code large language model via universal code. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1812–1824, Bangkok, Thailand. Association for Computational Linguistics.

Yu Sun, Xingyu Qian, Weiwen Xu, Hao Zhang, Chenghao Xiao, Long Li, Deli Zhao, Wenbing Huang, Tingyang Xu, Qifeng Bai, and Yu Rong. 2025. ReasonMed: A 370K multi-agent generated dataset for advancing medical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26446–26467, Suzhou, China. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2025a. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024. Planning in natural language improves llm search for code generation. *Preprint*, arXiv:2409.03733.

Jikai Wang, Juntao Li, Jianye Hou, Bowen Yan, Lijun Wu, and Min Zhang. 2025b. Efficient reasoning for llms through speculative chain-of-thought. *Preprint*, arXiv:2504.19095.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

Yihan Wang, Peiyu Liu, and Xin Yang. 2025c. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. *Preprint*, arXiv:2503.18596.

Yingce Xia, Xu Tan, Fei Tian, Tao Qin, Nenghai Yu, and Tie-Yan Liu. 2018. Model-level dual learning. In *International Conference on Machine Learning*, pages 5383–5392. PMLR.

Dongge Xue, Zhili Pu, Zhentao Xia, Hongli Sun, Ruihui Hou, Guangya Yu, Yupian Lin, Yongqi Fan, Jingping Liu, and Tong Ruan. 2025. Text-to-ES bench: A comprehensive benchmark for converting natural language to Elasticsearch query. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19767–19790, Vienna, Austria. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. Association for Computational Linguistics.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. 2025. Optimizing reasoning for text-to-SQL with execution feedback. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19206–19218, Vienna, Austria. Association for Computational Linguistics.

Ruichen Zhang, Rana Muhammad Shahroz Khan, Zhen Tan, Dawei Li, Song Wang, and Tianlong Chen. 2025. The quest for efficient reasoning: A data-centric benchmark to cot distillation. *arXiv preprint arXiv:2505.18759*.

Wenqing Zheng, S P Sharan, Ajay Kumar Jaiswal, Kevin Wang, Yihan Xi, Dejia Xu, and Zhangyang Wang. 2023. Outline, then details: Syntactically guided coarse-to-fine code generation. *Preprint*, arXiv:2305.00909.

A Data Construction Details

A.1 Detail validation for TERD

we argue that complex Elasticsearch (ES) reasoning data should be constructed following a multi-step procedure: i) Rewrite and analyze the question; ii) Identify index-related information involved in the question; iii) Explore the mapping between user queries and the index schema; iv) Select appropriate ES query syntax keywords; v) Determine whether the query involves aggregation operations; vi) Define appropriate post-processing code logic; vii) Synthesize the complete reasoning flow. The corresponding prompts are shown in Figure 8, Figure 9.

A.2 Detail validation for ETRD

The generated natural language description must meet the following quality standards: (i) it should clearly mirror the structure and components of the original Elasticsearch query, allowing for easy identification and mapping back to specific query elements; (ii) it must comprehensively cover all key components including filters, aggregations, sorting, and field selections without omission; (iii) the language should be easy to understand and follow, utilizing precise and appropriate terminology to make the query intent accessible; (iv) the description must maintain logical consistency, with each part transitioning smoothly to the next, free of contradictions or abrupt shifts; (v) all parameters, values, ranges, and constraints from the original query must be represented exactly as they are, without any misinterpretation or alteration; (vi) if the query includes aggregations, they must be correctly interpreted and explained in clear, natural language; (vii) the description should retain the complete semantic meaning and context of the original query, ensuring that its intent is fully and accurately conveyed throughout the conversion. The corresponding prompts are shown in Figure 10.

A.3 Data annotation detail graph

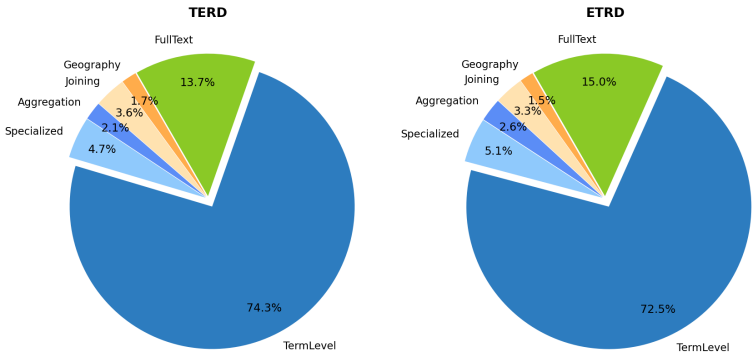

Figure 6: Percentage distribution chart of different categories.

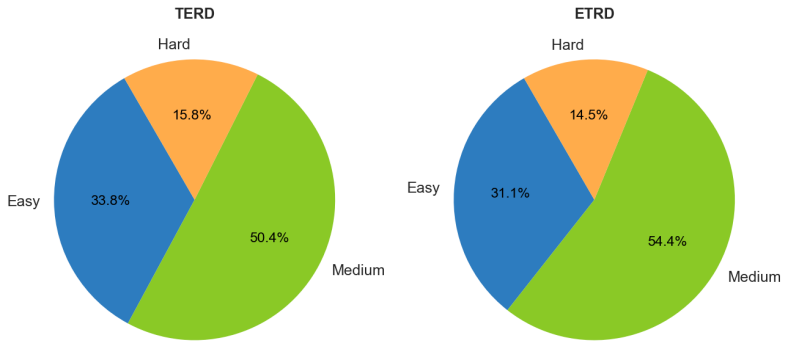

Figure 7: Percentage distribution chart of different difficulty.

B Supplement for Dual-CoRE

B.1 Case of Dual-CoRE

Table 4: Case Study: Complete DualES Reasoning with Step-by-Step Thought and Initial Output

Category	*Content*
Question	Please specify all of the schools and their related mailing zip codes that are under Avetik Atoian's administration.
Schema	`Index: california_schools_schools` `Fields: School, MailZip, AdmFName1, AdmLName1, AdmFName2, AdmLName2, AdmFName3, AdmLName3...`

Text-to-ES Reason	**Reasoning:**
	1. Problem Analysis: Identify schools and mailing zip codes under "Avetik Atoian". Requires matching admin name and returning `School` and `MailZip`.
	2. Determine Index: The relevant index is `california_schools_schools` because it contains administrator names, school name, and mailing zip code.
	3. Field Mapping: Match either: `AdmFName1 + AdmLName1`, `AdmFName2 + AdmLName2`, or `AdmFName3 + AdmLName3`.
	4. Query Type Selection: Use a `bool` query with `should` clauses for name combinations.
	5. Aggregation Needs: None. We need raw documents.
	6. Result Post-processing: Extract `School` and `MailZip` from each hit.
	7. Query Logic: Build a `bool` query with `should` clauses for each combination.
	ESCode: `# Initial Answer Output`
	`query = { "query": { "bool": { "should": [`
	`  { "bool": { "must": [ {"term": {"AdmFName1.keyword": "Avetik"}}, {"term":`
	`{"AdmLName1.keyword": "Atoian"}} ] } },`
	`  { "bool": { "must": [ {"term": {"AdmFName2.keyword": "Avetik"}}, {"term":`
	`{"AdmLName2.keyword": "Atoian"}} ] } },`
	`  { "bool": { "must": [ {"term": {"AdmFName3.keyword": "Avetik"}}, {"term":`
	`{"AdmLName3.keyword": "Atoian"}} ] } }`
	`], "minimum_should_match": 1 } }, "_source": ["School", "MailZip"] }`
	`response = esclient.search(index="california_schools_schools", body=query, size=1000)`
	`# Extract results`
	`result = []; for hit in response['hits']['hits']:`
	`  source = hit['_source']; school = source.get('School', '');`
	`  mail_zip = source.get('MailZip', ''); result.append((school, mail_zip))`

ES-to-Text Reason	**Reasoning:** The index schema indicates that the query is targeting the `california_schools_schools` index. The query uses a boolean `should` clause with three `must` conditions, each checking for an exact match on the administrator's names across `AdmFName1`, `AdmLName1`; `AdmFName2`, `AdmLName2`; and `AdmFName3`, `AdmLName3`. The `minimum_should_match` of 1 means at least one condition must be true. The query requests `School` and `MailZip`. Therefore, the query aims to find schools where at least one administrator named Avetik Atoian is listed. **Generated Question**: What are the names and mailing zip codes of schools that have an administrator named Avetik Atoian, regardless of whether they are listed as the first, second, or third administrator?
Format Rethink	The model confirmed adherence to the step-by-step reasoning and output constraints.
Content Rethink	**Reason:** Both questions seek information about schools for Avetik Atoian. However, the original question specifies schools "under his administration," implying he is the primary administrator. The generated question broadens this to any of the three administrator slots. Since the original focuses on primary administration and the generated includes any position, the filters are not equivalent. **Judge**: DIFFERENT - The original question specifies schools under administration, while the generated includes schools where he is listed in any slot, broadening the scope.

Final EScode	`query = { "query": { "bool": { "must": [`
	`  {"term": {"AdmFName1.keyword": "Avetik"}},`
	`  {"term": {"AdmLName1.keyword": "Atoian"}}`
	`] } }, "_source": ["School", "MailZip"] }`
	`for doc in scan(client=esclient, index=index_name, query=query):`
	`  source = doc["_source"]`
	`  result.append((source["School"], source["MailZip"]))`

Golden ES Code	`query = { "query": { "bool": { "must": [`
	`  {"term": {"AdmFName1.keyword": "Avetik"}},`
	`  {"term": {"AdmLName1.keyword": "Atoian"}}`
	`] } }, "_source": ["School", "MailZip"] }`
	`index_name = "california_schools_schools"`
	`for doc in scan(client=esclient, index=index_name, query=query):`
	`  source = doc["_source"]`
	`  result.append((source["School"], source["MailZip"]))`

B.2 Pseudo Code of Dual-CoRE

Algorithm 1 Dual-CoRE Framework

Require: Query Q
Ensure: Optimized ES Query ES_{final}

```
feedback ← ∅
Steps ← {Analysis, Index, Mapping, Type, Aggregation, Post-proc, Logic}
repeat
    (Reason, ES) ← ModelA(Q + feedback)                    // Stage 1: Reasoning
    if Format is invalid or Reason misses any s ∈ Steps then
        Miss ← Steps \ IdentifySteps(Reason)
        feedback ← "Refine reasoning process, missing: " + Miss
        continue
    else
        T_gen ← ModelB(ES)                                 // Stage 2: Result Reflection
        if Compare(T_gen, Q) = SAME then
            return ES
        else
            feedback ← "Discrepancy detected: " + Diff(T_gen, Q)
        end if
    end if
until reached max_attempts
```

C Experiment Details

C.1 Training Details

For model training, we utilize Swift. For training on our datasets, we adopt the LoRA method with the following hyperparameters: number of epochs = 1, learning rate = 1e-4, batch size = 64, rank $r = 64$, and alpha $\alpha = 256$. The base models include Qwen2.5-7B-Instruct, Qwen2.5-Coder-7B-Instruct, Qwen2.5-14B-Instruct, Qwen2.5-Coder-14B-Instruct, Qwen3-8B, and Qwen3-14B, which are fine-tuned using four strategies: Dual-FD, Dual-BD, Dual-MT, and Dual-CoRE. All models are trained on a cluster of eight NVIDIA A100 (80GB) GPUs.

C.2 Inference Details

For local model inference, we accelerate LLM serving using vLLM. For API-based inference, we utilize the APIs from for All the Advanced Models in Table 2. In all experiments, we set the temperature to 0.0001 and the maximum token limit to 7500. The detailed prompt can be found in Figure 13.

D Other Experiments

Models	LED			BirdES		
	DSLEM	EX	VES	DSLEM	EX	VES
Zero-shot						
MiniMax-M2.5	1.56	25.47	26.92	1.39	17.55	18.69
GPT-4o-mini	6.10	33.46	35.01	1.47	9.04	9.14
GPT-4.1-Nano	2.53	34.04	35.31	1.62	16.00	16.52
Qwen3-max-preview	4.28	32.48	34.21	4.12	40.82	41.32
Deepseek-V3.1	3.63	27.29	27.93	4.42	30.61	31.78
Deepseek-V3.2	4.42	30.60	32.30	3.38	27.02	27.66
Deepseek-Reasoner	2.14	28.07	29.54	4.56	34.43	34.85
One-shot						
MiniMax-M2.5	22.08	44.19	44.20	3.01	25.55	25.65
GPT-4o-mini	21.45	43.87	44.09	5.13	27.01	27.24
GPT-4.1-Nano	27.42	60.86	62.17	3.01	23.26	23.48
Qwen3-max-preview	29.04	61.02	61.39	8.81	48.97	50.27
Deepseek-V3.1	25.53	55.04	56.35	5.36	36.12	36.71
Deepseek-V3.2	26.97	59.84	60.58	4.04	41.77	42.27
Deepseek-Reasoner	22.15	61.71	62.60	5.07	46.39	46.94

Table 5: Zero-shot vs One-shot performance improvement.

E Prompt Examples

Instruction for Text-to-ReasonES(Human-Assigned)

You are an Elasticsearch query expert. Your task is to analyze a natural language question and derive the correct Elasticsearch query and post-processing code through a step-by-step reasoning process — without relying on any provided example code.

Important Guidelines:
- Do NOT reference or copy any example code — reason from first principles.
- Your reasoning must be realistic and reflect how an expert would approach the problem.
- The final code must be syntactically correct and functionally equivalent to the expected solution.
- Use only the fields and structure defined in the provided index schema.

Follow this structured reasoning process:
1.Problem Analysis: Understand what the user is asking for — identify the core intent and desired outcome. If the problem is complex, break it down into multiple sub-problems.
2.Determine Index: Based on the problem analysis, determine which index name needs to be queried. Also determine whether cross-index queries are required.
3.Field Mapping: Determine which Elasticsearch fields are relevant based on the schema and question context.
4.Query Type Selection: Decide the appropriate query structure (e.g., match, term, range, bool, nested, etc.).
5.Aggregation Needs: Determine if aggregation (e.g., terms, stats, date_histogram) is required to summarize or group results.
6.Result Post-processing: Plan how to transform the raw ES response into the expected output format (e.g., extracting values, flattening nested structures).
7.Query Logic: Outline the general logic of the query code — a brief pseudocode snippet can be provided.
Other important guidelines:
Elasticsearch query code and post-processing
Use the pre-defined 'esclient' directly — do not redefine it.
Store the final result in a tuple variable named 'result'.
Do not include any print() statements.
Elasticsearch Index Schema:
{item.get('index_mapping', '')}

Elasticsearch Field Descriptions:
{item.get('field_descriptions', '')}

Natural Language Question:
{item.get(question, '')

ES Query Code:
{item.get('query', '')}

Output format:
<think>...think process...</think><answer>...final code...</answer>

Figure 8: Construction prompt for the TERD dataset(Human-Assigned).

Instruction for Text-to-ReasonES-(LLM-Generated)

You are an Elasticsearch query expert. Your task is to analyze a natural language question and derive the correct Elasticsearch query and post-processing code through a step-by-step reasoning process — without relying on any provided example code.

Important Guidelines:
- Do NOT reference or copy any example code — reason from first principles.
- Your reasoning must be realistic and reflect how an expert would approach the problem.
- The final code must be syntactically correct and functionally equivalent to the expected solution.
- Use only the fields and structure defined in the provided index schema.

Follow this structured reasoning process:
1. Problem Analysis: Understand what the user is asking for — identify the core intent and desired outcome.
2. Field Mapping: Determine which Elasticsearch fields are relevant based on the schema and question context.
3. Query Type Selection: Decide the appropriate query structure (e.g., match, term, range, bool, nested, etc.).
4. Filtering & Conditions: Identify all filtering criteria, including exact matches, ranges, date constraints, or boolean logic.
5. Aggregation Needs: Determine if aggregation (e.g., terms, stats, date_histogram) is required to summarize or group results.
6. Result Post-processing: Plan how to transform the raw ES response into the expected output format (e.g., extracting values, flattening nested structures).

Other important guidelines:
Elasticsearch query code and post-processing
Use the pre-defined 'esclient' directly — do not redefine it.
Store the final result in a tuple variable named 'result'.
Do not include any print() statements.
Elasticsearch Index Schema:
{item.get('index_mapping', '')}

Elasticsearch Field Descriptions:
{item.get('field_descriptions', '')}

Natural Language Question:
{item.get(question, '')

ES Query Code:
{item.get('query', '')}

Output format:
<think>...think process...</think><answer>...final code...</answer>

Figure 9: Construction prompt for the TERD dataset(LLM-Generated).

Instruction for ES-to-ReasonText

You are an Elasticsearch query expert. Your task is to analyze an Elasticsearch query (including its index schema and query body) and generate a precise, fluent natural language question that captures the exact intent of the query—through a step-by-step reasoning process, without relying on any provided example question.

Important Guidelines:
- Do NOT reference, mimic, or reproduce the provided natural language question.
- Reason from first principles using only the index schema and the actual query logic.
- The generated question must be semantically equivalent to the Elasticsearch query: every filter, range, match, and geographic condition must be accurately reflected.
- Use clear, domain-appropriate terminology based on the field descriptions (e.g., “mass in grams” not just “mass”).
- The question should specify what information is to be returned, based on the `_source` fields in the query.

Follow this structured reasoning process:
1. Index Schema Understanding: Examine the index mapping and field descriptions to understand the meaning, data type, and domain context of each relevant field (e.g., 'geolocation' is a geo_point representing latitude/longitude; 'mass' is in grams).
2. Query Structure Analysis: Identify the top-level query type (e.g., bool, match_all) and decompose its clauses (must, filter, should, etc.).
3. Condition Interpretation: Translate each query clause into its semantic intent:
  - A `match` or `term` on a keyword-like field → exact categorical match (e.g., class = "Iron, IIE-an")
  - A `range` query → numerical or temporal constraint (e.g., mass < X grams, year after Y)
  - A `geo_bounding_box` → spatial constraint within a rectangle defined by two corners
4. Output Specification Analysis: Inspect the `_source` field list to determine which attributes should be included in the result, and reflect this in the natural language question (e.g., “list the name, mass, and geolocation”).
5. Synthesis: Combine all interpreted conditions and output requirements into a single, coherent, grammatically correct natural language question that fully represents the query’s intent.

Elasticsearch Index Schema:
{item.get('index_mapping', '')}

Elasticsearch Field Descriptions:
{item.get('field_descriptions', '')}

Elasticsearch Query Code:
{item.get('query', '')}

Natural Language Question:
{item.get(Question, '') }

Output Format:
<think> think process</think><answer>natural language question</answer>

Figure 10: Construction prompt for the ETRD dataset.

Instruction for Difficulty Classification

You are an expert Elasticsearch query evaluator specializing in difficulty assessment. Your task is to analyze the complexity of Elasticsearch queries based on their structural complexity, logical reasoning requirements, and index interaction patterns.

Difficulty Classification Criteria:

Easy: Simple queries with minimal conditions and straightforward logic
- Single index operations only
- Basic query types (match, term, range)
- No nested queries or complex boolean combinations
- Direct field-to-value mappings

Medium: Queries with moderate complexity requiring logical reasoning
- Multi-condition queries with boolean logic (AND/OR/NOT)
- Single index with nested query structures
- Involves exactly two indices with simple joins/associations
- Basic aggregations or sorting requirements

Hard: Complex queries requiring advanced reasoning and optimization
- Involves three or more indices
- Complex nested query structures with multiple levels
- Advanced aggregations, subqueries, or cross-index operations
- Requires sophisticated data modeling and query optimization

Analysis Task:
Evaluate the difficulty level of the following Elasticsearch query implementation:

Index Schema:
{item.get('index_mapping', '')}

User Question:
{item.get(Question, '') }

Generated Solution (Reasoning + Code):
{item.get('prompt_answer', '')}

Respond with exactly one difficulty level: Easy, Medium, or Hard.

Figure 11: Difficulty classification prompt.

Instruction for Score

You are an Elasticsearch query evaluation expert. Your task is to evaluate the quality of a generated Elasticsearch query and reasoning process based on the given criteria.

Evaluation Criteria (1-8 scale, where 8 is perfect):
1. **Syntax Correctness**: Code has no syntax errors, proper structure and formatting
2. **Logic Correctness**: Query logic correctly addresses the user's question
3. **Reasoning Quality**: Step-by-step analysis is thorough and expert-level
4. **Schema Compliance**: Proper use of index schema, correct field types (especially keyword vs text fields), no invalid field references
5. **Query Appropriateness**: Query type selection matches the task requirements
6. **Result Processing**: Post-processing correctly transforms results to expected format
7. **Efficiency**: Query is optimized, avoids unnecessary complexity
8. **Completeness**: Solution fully addresses all aspects of the user's question

Elasticsearch Index Schema:
{item.get('index_mapping', '')}

Natural Language Question:
{item.get(question, '')}

Expected Correct ES Query:
{item.get('query', '')}

Generated Reasoning and Code:
{item.get('prompt_answer', '')}

Output only a single number between 1.0 and 8.0 representing the overall quality score:

Figure 12: Data quality scoring prompt.

Instruction for Text-to-ESReason

Follow this structured reasoning process:
1.Problem Analysis: Understand what the user is asking for — identify the core intent and desired outcome. If the problem is complex, break it down into multiple sub-problems.
2.Determine Index: Based on the problem analysis, determine which index name needs to be queried. Also determine whether cross-index queries are required.
3.Field Mapping: Determine which Elasticsearch fields are relevant based on the schema and question context.
4.Query Type Selection: Decide the appropriate query structure (e.g., match, term, range, bool, nested, etc.).
5.Aggregation Needs: Determine if aggregation (e.g., terms, stats, date_histogram) is required to summarize or group results.
6.Result Post-processing: Plan how to transform the raw ES response into the expected output format (e.g., extracting values, flattening nested structures).
7.Query Logic: Outline the general logic of the query code — a brief pseudocode snippet can be provided.

Relevant Examples:
..

Input
INDICES DESCRIPTION:
{indices_desc}
Fields Description:
{fields_desc}
QUESTION:{question}
SUPPLEMETN:{improve}
ANSWER:

Output format
<think>...reasoning process...</think><answer>...final code...</answer>

Figure 13: Text-to-ESReason prompt.

Instruction for ES-to-TextReason

You are an Elasticsearch query expert. Your task is to analyze an Elasticsearch query (including its index schema and query body) and generate a precise, fluent natural language question that captures the exact intent of the query—through a step-by-step reasoning process, without relying on any provided example question.
Important Guidelines:
- Do NOT reference, mimic, or reproduce the provided natural language question.
- Reason from first principles using only the index schema and the actual query logic.
- The generated question must be semantically equivalent to the Elasticsearch query: every filter, range, match, and geographic condition must be accurately reflected.
- Use clear, domain-appropriate terminology based on the field descriptions (e.g., "mass in grams" not just "mass").
- The question should specify what information is to be returned, based on the `_source` fields in the query.

Follow this structured reasoning process:
1. Index Schema Understanding: Examine the index mapping and field descriptions to understand the meaning, data type, and domain context of each relevant field.
2. Query Structure Analysis: Identify the top-level query type (e.g., bool, match_all) and decompose its clauses (must, filter, should, etc.).
3. Condition Interpretation: Translate each query clause into its semantic intent:
  - A `match` or `term` on a keyword-like field → exact categorical match
  - A `range` query → numerical or temporal constraint
  - A `geo_bounding_box` → spatial constraint within a rectangle defined by two corners
4. Output Specification Analysis: Inspect the `_source` field list to determine which attributes should be included in the result, and reflect this in the natural language question.
5. Synthesis: Combine all interpreted conditions and output requirements into a single, coherent, grammatically correct natural language question that fully represents the query's intent.
Elasticsearch Index Schema:
{index_mapping}

Elasticsearch Query Code:
{query}

Output Format:
<think> Thinking process</think><answer>natural language question</answer>

Figure 14: ES-to-TextReason prompt.

Instruction for TERD Verification

You are an expert evaluator for Text-to-ES (natural language to Elasticsearch query) systems. Your task is to judge whether the reasoning process provided for converting the user question into an ES query is acceptable. Evaluate only the reasoning process (the step-by-step analysis), not the final code correctness.

Consider the following criteria when judging the reasoning process:
- Problem Analysis: The reasoning should identify what the user is asking for — core intent and desired outcome; if the problem is complex, it should break it down into sub-problems.
- Determine Index: Based on the problem analysis, the reasoning should determine which index name(s) to query and whether cross-index queries are required.
- Field Mapping: The reasoning should determine which Elasticsearch fields are relevant based on the schema and question context.
- Query Type Selection: The reasoning should decide the appropriate query structure (e.g., match, term, range, bool, nested, etc.).
- Aggregation Needs: The reasoning should determine if aggregation (e.g., terms, stats, date_histogram) is required to summarize or group results.
- Result Post-processing: The reasoning should plan how to transform the raw ES response into the expected output format (e.g., extracting values, flattening nested structures).
- Query Logic: The reasoning should outline the general logic of the query — e.g. a brief pseudocode or step-by-step logic of how the query will be built and executed.

If the reasoning process meets these criteria overall (where applicable to the question), answer KEEP. If it has serious problems (e.g. major missing steps, vague or inconsistent reasoning, failure to cover key aspects like index/field/query-type), answer DISCARD.

Reply in the following JSON format only, with no other text. The "decision" field must be exactly either "KEEP" or "DISCARD":
{{"decision": "KEEP", "reason": "<brief one-sentence reason>"}}

Natural Language Question:
{item.get(question, '')}

Index Schema:
{item.get('index_mapping', '')}

Field Descriptions:
{item.get('field_descriptions', '')}

Golden ESCode:
{item.get('query', '')}

Reasoning Process to Evaluate:
{reasoning_text}

Figure 15: TERD evaluation prompt.

Instruction for ETRD Verification

You are an expert evaluator for Elasticsearch query to natural language conversion systems. Your task is to judge whether the conversion from Elasticsearch query to natural language text is acceptable.

Consider the following criteria when judging:
- The generated text should clearly reflect the structure and key components of the original Elasticsearch query (traceability).
- It should cover filters, aggregations, sorting, field selections without omitting important components (coverage).
- It should be easy to understand with clear terminology (comprehensibility).
- It should be logically consistent without contradictions (logical coherence).
- Parameters, values, ranges, and constraints should be accurately represented (parameter accuracy).
- Aggregations, if any, should be correctly interpreted in natural language (aggregation interpretation).
- The complete semantic meaning and context of the original query should be preserved (context integrity).

If the conversion meets these criteria overall, answer KEEP. If it has serious problems (e.g. major omissions, wrong parameters, incoherent or misleading description), answer DISCARD.

Reply in the following JSON format only, with no other text. The "decision" field must be exactly either "KEEP" or "DISCARD":
{{"decision": "KEEP", "reason": "<brief one-sentence reason>"}}

Original Elasticsearch Query:
{item.get('query', '')}

Index Schema:
{item.get('index_mapping', '')}

Field Descriptions:
{item.get('field_descriptions', '')}

Golden Quesition
{item.get(question, '')}

Generated Natural Language to Evaluate:
{reasoning_text}

Figure 16: ETRD evaluation prompt.