DERIVATION-SQL: A COST-EFFECTIVE GRAPH-BASED FRAMEWORK FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SQL enables users to query relational databases in natural language and has broad applications. However, existing retrieval approaches based on question semantics may ignore the derivation relationships among SQL queries. In contrast, structure-based approaches that use predicted SQL require an additional generation step, increasing inference cost. To address these issues, we propose Derivation-SQL, a cost-effective Text-to-SQL framework that uses SQL derivation graphs to retrieve structurally relevant examples directly from natural language questions. We first construct an SQL derivation graph (DG) by decomposing training SQL queries into subqueries and connecting queries through derivation relationships. This process also expands the pool of few-shot candidates. We then design a DG-based embedding network (DGE) that encodes text semantics and SQL derivation relationships. The network encodes examples offline and maps natural language questions to the same embedding space online. A tailored pipeline with improved prompts seamlessly integrates these components into an end-to-end solution. Experiments on BIRD and Spider show that Derivation-SQL achieves competitive execution accuracy with low token consumption, reaching a Pareto-optimal balance between accuracy and cost. With GPT-4o, Derivation-SQL achieves execution accuracy of 72.56% on BIRD-Dev, which costs less than half the token usage of other representative approaches.

1 INTRODUCTION

Text-to-SQL refers to the process of converting Natural Language (NL) questions into SQL queries. By bridging the gap between non-expert users and database access, it enables non-technical users to explore insights directly from databases and boosts work efficiency for specialists like data analysts. It has wide potential applications, including business intelligence, ad-hoc data analysis, healthcare administration, and so on (Hong et al., 2024; Liu et al., 2025a; Shi et al., 2026). The importance of Text-to-SQL is highlighted by the 2023 Stack Overflow survey, which shows that over half (51.52%) of professional developers utilize SQL in their work, yet only around one-third (35.29%) have received systematic training in it (Hong et al., 2024).

Recently, benefiting from the development of Large Language Model (LLM) techniques, Text-to-SQL has made significant advancements. For example, execution accuracy on BIRD[1] benchmark is from 7.06 to 81.67 between February 2023 and December 2025. Building on foundational LLMs, progress has been enhanced by prompt engineering (e.g., few-shot learning and chain-of-thought reasoning) (Dong et al., 2023; Wei et al., 2022; Pourreza & Rafiei, 2023; Xie et al., 2025) and fine-tuning methods (e.g., LoRA) (Gu et al., 2022; Li et al., 2023b). From whom, many approaches provide few-shot examples to assistant LLM to generate SQL queries. By grounding generation in retrieved, actionable context, these approaches provide the necessary dynamism and efficiency for complex, real-world databases. Researchers explore to use semantic similarity (Li et al., 2025a), SQL-structure similarity (Gao et al., 2024; Xie et al., 2025) and AST-based similarity (Shen et al., 2024) to find out the relevant few-shot examples to the asked questions. Text semantic and SQL-structure approaches both rely on language models (LMs) to encode questions and SQL queries. The latter, along with AST-based methods, often pre-generate SQL queries and compare them to few-shot examples, increasing computational cost. Since semantic similarity are not only more efficient

[1] https://bird-bench.github.io/

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

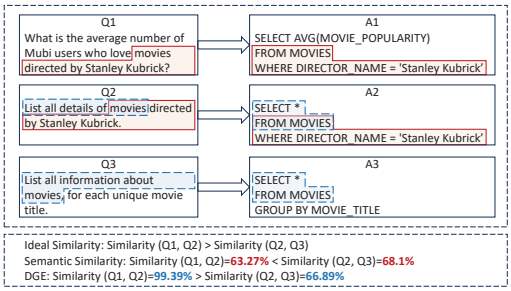

Figure 1: Failure of semantic similarity in Text-to-SQL. Red and blue mark overlapping content in (Q1, Q2; A1, A2) and (Q2, Q3; A2, A3), respectively. Ideally, the similarity between A1 and A2 should be *higher* than that between A2 and A3, since A1 is an extension of A2, while A2 and A3 cannot be derived from each other and thus should exhibit low similarity. However, semantic similarity measures commonly used in Text-to-SQL methods (Li et al., 2025a; Xie et al., 2025) produce the opposite results. Our Derivation Graph Embedding Network (DGE) directly yields the correct similarity by involving SQL derivation.

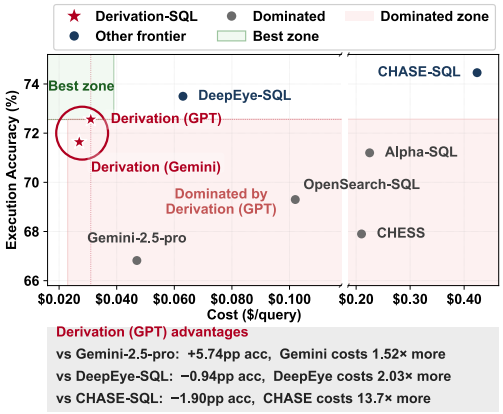

Figure 2: Cost–performance analysis on BIRD-Dev set. Each point denotes a method; the x-axis is the average LLM cost per question (USD/query), and the y-axis is execution accuracy (EX). The horizontal axis is broken to accommodate the large cost gap between efficient and expensive approaches. Derivation-SQL (GPT) achieves strong EX at low cost; every baseline with higher EX than ours consumes at least 2× as many tokens.

but also more commonly used than SQL-structure or AST-based similarities, we concentrate our comparative analysis on DGE and semantic similarity, rather than the less efficient alternatives.

Existing semantic-based approaches still face several challenges: *(i) Text similarity Limitations.* Most approaches (Liu et al., 2025b; Xie et al., 2025) encode examples using text semantic models (e.g., Bert) and then to measure the correlation between NL questions of examples, failing to capture and exploit the richer, structural correlations among SQL queries themselves. *(ii) Limited Few-shot Candidates.* A common challenge for many approaches is the limited availability and diversity of suitable few-shot candidates. Existing methods select few-shot candidates from various sources, such as fixed examples (often infeasible) (Talaei et al., 2024), the training set (limiting size and diversity) (Xie et al., 2025), external data (not always available) (Li et al., 2025d), or LLM-generated examples (which may be noisy or unrealistic) (Pourreza et al., 2025a).

Example 1 *Fig. 1 demonstrates the limitations of text semantic similarity in the Text-to-SQL task. Semantic similarity does not fully meet the requirements of Text-to-SQL, which is the main challenge and motivation of this research. For semantic similarity, Q2 and Q3 are closer than Q1 and Q2. However, from the Text-to-SQL perspective, A1 and A2 are more correlated, as A1 can be seen as an extension of A2. This indicates that relying solely on text semantic similarity is insufficient for Text-to-SQL, and highlights the need for models better suited to SQL generation.*

In this paper, we aim to fulfill three aspirations: (i) The proposed approach provides LLMs with fine-grained, task-Relevant few-shot candidates. (ii) The proposed approach captures the semantic and derivation correlation among SQL queries, to tackle the limitations of existing few-shot-based approaches that rely solely on textual similarity for example selection. (iii) The proposed approach utilizes as few resources as possible to obtain excellent performance of Text-to-SQL. We propose Derivation-SQL, a cost-effective graph-based framework that explicitly models SQL Derivation for Text-to-SQL (Derivation-SQL). First, we construct an innovative SQL Derivation Graph (DG), where each node represents a specific example. To this end, we decompose SQL queries into sub-queries based on their derivation relationships. This decomposition not only increases the diversity and size of few-shot candidates, but also allows us to focus on the parts that are most relevant to the target problem, thereby avoiding interference from unrelated sections. Next, we propose an embedding network SQL Derivation Graph Embedding network (DGE). DGE encodes NL questions by modeling the derivation relationship among SQL queries. Finally, we optimize prompts and design dedicated adaptation modules, making DG and DGE apply to Text-to-SQL better and trying

to reduce the computing resources. By conducting experiments, we verify that the performance of Derivation-SQL is remarkable in terms of resource utilization. Fig. 2 compares the performance and cost of various Text-to-SQL approaches. The proposed Derivation-SQL achieves excellent performance with the lowest overall cost.

In summary, our contributions are four-fold: (1) We propose Derivation-SQL, a robust and cost-effective derivation graph-based framework for Text-to-SQL. (2) We design the novel SQL derivation graph (DG). By decomposing SQL into sub-queries for graph construction, the DG provides richer, structural insights for accurate SQL generation. (3) We design the DG-based embedding network (DGE), which uniquely jointly models textual semantics and SQL relational structures during example encoding. This ensures the retrieval of context with more precise guidance compared to semantic-only matching. (4) By extensive experiments, we demonstrate that Derivation-SQL achieves the lowest computational cost among models with comparable performance on benchmark datasets, while maintaining highly competitive results.

2 RELATED WORK

Recent Text-to-SQL research has largely followed two paths: In-Context Learning (ICL) or Fine-Tuning (FT). ICL approaches design sophisticated prompts using strategies like Chain-of-Thought (CoT) (Wei et al., 2022; Xie et al., 2025) to generate intermediate reasoning steps, or Divide-and-Conquer (Pourreza & Rafiei, 2023; Li et al., 2025a) to break down complex questions. FT-based methods (Gu et al., 2023; Liu et al., 2025b) adapt pre-trained LLMs to the Text-to-SQL task for improved performance. Both ICL and FT methods (Shen et al., 2024; Gao et al., 2024; Liao et al., 2026) often adopt few-shot learning to enhance generation by retrieving relevant examples. Most works retrieve examples from the original training set using efficient Approximate Nearest Neighbor Search (ANNS). Others explore more complex retrieval, such as using dedicated ranking models (Li et al., 2025d) or leveraging LLMs to synthesize new examples from external data (Cai et al., 2025), often at a higher computational cost. Different from these works, our method, Derivation-SQL, focuses on improving the retrieval process itself. While compatible with foundational techniques like Divide-and-Conquer, Derivation-SQL enhances example diversity by mining new, meaningful SQL queries directly from the training set without relying on external information or LLM-based synthesis. This allows us to improve the quality of few-shot examples while maintaining the high efficiency of online ANNS-based retrieval. Moreover, many works have explored methods to decompose SQL queries, such as STEPS (Tian et al., 2023), PARSQL (Dai et al., 2025), and LEARNAT (Liao et al., 2026), but they have limitations in handling complex queries, generating synthetic sub-queries, or maintaining efficiency. In contrast, Derivation-SQL designs a heuristic approach that efficiently extracts SQL derivation relationships and handles highly complex SQL queries.

3 PROBLEM FORMULATION

Key Notations. Let q denote a NL question. Its ground-truth SQL query, which conforms to SQL standards and correctly answers q, is denoted by s. Let s' denote the SQL query generated by our Derivation-SQL framework. The database schema is denoted as $\mathcal{S} = \{(D_i, \mathcal{T}_i, \mathcal{C}_i, \mathcal{V}_i)\}_{i=1}^{N_s}$, where each tuple describes one database (detailed description in Appendix B) and N_s is the number of databases. We define the training set as $\mathcal{D}_t = \{(q_i, s_i)\}_{i=1}^{N_t}$, where q_i is an NL question and s_i is its corresponding SQL query. Here $\mathcal{D}_t$ is also the set of example candidates.

Derivation-SQL Problem. Given an NL question q, database schema $\mathcal{S}$, and the training set $\mathcal{D}_t$, the goal of Derivation-SQL problem is to minimize the discrepancy between the generated SQL s' and the ground-truth s by optimizing the prompt $\mathbf{P}$. Formally, this is formulated as

$$\arg\min_{\mathbf{P}} \mathcal{L}(s', s \mid q, \mathcal{S}, \mathcal{D}_t), \tag{1}$$

where $\mathcal{L}$ denotes the loss function of Derivation-SQL. Given the NL query q, the predicted s' depends on q, the schema $\mathcal{S}$, a set of examples retrieved from $\mathcal{D}_t$, together with an optimized prompt $\mathbf{P}$, while s provides the supervision signal.

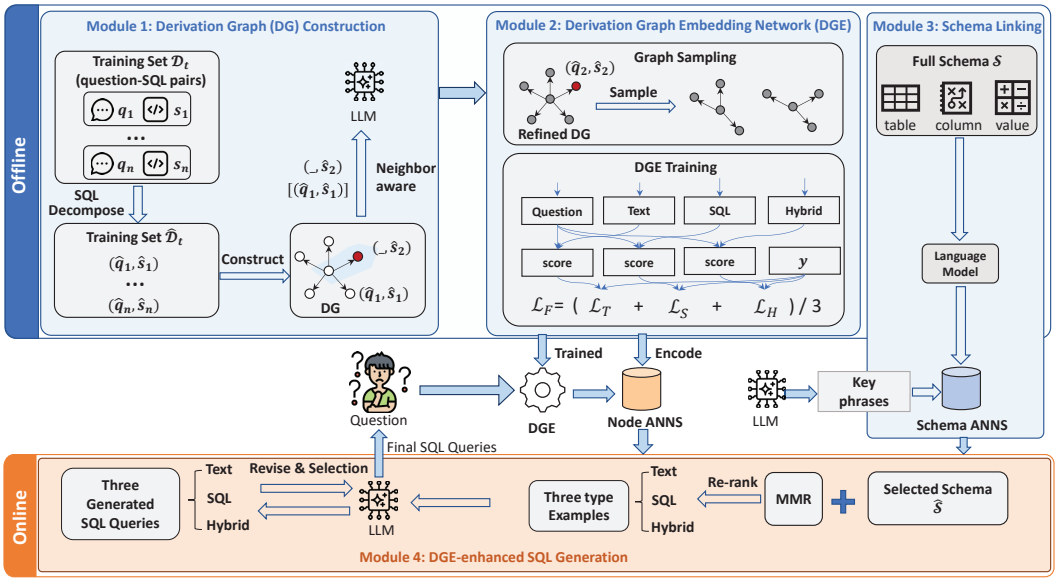

Figure 3: The architecture of Derivation-SQL. Derivation-SQL consists of four key modules: **Module 1: DG Construction.** This module decomposes SQL queries, constructs DG, and generates questions for the corresponding nodes. **Module 2: DGE.** This module details the training process of DGE as well as its usage. After training, DGE is utilized to encode all nodes and is further used for online question encoding during inference. **Module 3: Schema Linking.** This module aims to encode the schema information offline and store it within an ANNS structure, enabling efficient schema retrieval during online inference. **Module 4: DGE-enbanced SQL Generation.** This module focuses on generating SQL queries by leveraging the retrieved examples obtained through DGE.

4 METHODOLOGY

The framework of Derivation-SQL is shown in Fig. 3. We introduce the main modules DG and DGE below. The rest modules are introduced in the Appendix. A.

4.1 SQL DERIVATION GRAPH CONSTRUCTION

For previous work (Xie et al., 2025; Gao et al., 2024) only utilized few-shot candidates but overlooked to analyze the inner logic among them. That results in the limited availability and diversity of suitable few-shot candidates. As shown in Fig. 4, many SQL queries might share some common precursor SQL queries. Drawing inspiration from this finding, we propose the SQL Derivation Graph (DG). The key idea of the DG construction is to systematically extract derivation relationships between question-SQL pairs by treating each SQL query as an individual entity. Our findings indicate that the accuracy of similarity measurement among SQL queries can be considerably enhanced through the utilization of DG. Our strategy constructs a derivation graph DG from the training set D_t in three intertwined stages: (1) **SQL Decomposition**. We recursively strip SQL operations in reverse execution order to derive all syntactically valid sub-queries, augmenting the original training data. (2) **DG Construction.** Simultaneously, we build a directed acyclic graph where nodes represent unique SQL query with its question and edges indicate that a SQL query is obtained by adding one operation to a simpler sub-query. (3) **Question Generation.** An LLM generates natural language questions for nodes that lack them, first using a majority-neighbour heuristic for rich context, then completing the remaining nodes to ensure full coverage. A detailed description, including an evaluation of question quality and a comparison with decomposition approaches, can be found in the Appendix. C.

4.2 SQL DERIVATION GRAPH EMBEDDING NETWORK

The objective of DGE is to encode both NL questions users ask and nodes of DG. The architecture of DGE is shown in Fig. 5. To make example retrieval utilize multi-views information greater, DGE is designed to have three type node embeddings.

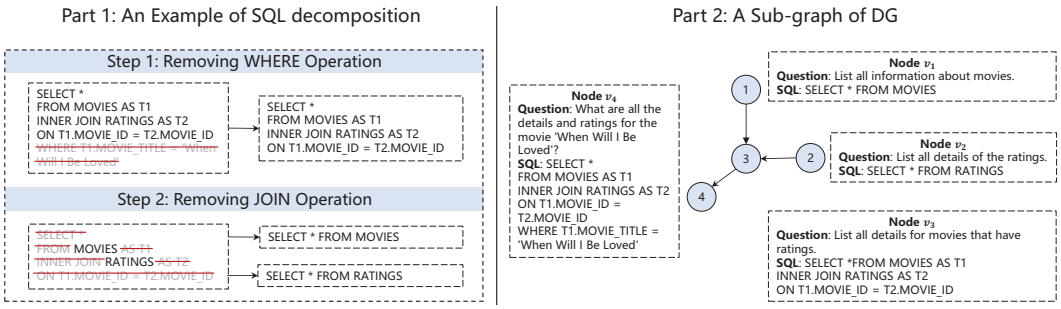

Figure 4: An example of DG Processing.

Shared Encoder. The shared encoder is used to encode textual information. We highlight it because it not only encodes the NL questions of nodes offline, but also encodes the NL questions that users ask online. First of all, encoding NL questions into token embeddings whose dimension is d_t by using Bert (all-MiniLM-L6-v2). For clarity, we use TextBert to denote the Bert model used here. Then, using the average pooling to aggregate the token embeddings. Specifically, we perform average pooling over the token embeddings from each of the last three layers of TextBert, obtaining three separate averaged embeddings $\{a_t^1, a_t^2, a_t^3\}$. Thirdly, a 2-layer Fully Connected Network (FCN) is adopted to map the three separate averaged embeddings to a d_t-dimensional vector e_t. It should be emphasized that GeLU (Hendrycks & Gimpel, 2016) and dropout (Srivastava et al., 2014) are adopted between any two layers of FCN.

$$e_t^b = \text{TextBert}(q), \quad a_t = \text{Pooling}(e_t^b), \quad c_t = \text{concat}(a_t^1, a_t^2, a_t^3), \quad e_t = \text{FCN}(c_t) \tag{2}$$

For user question encoding, we extra normalize the embedding e_q, i.e., $\hat{e}_q = \frac{e_t}{|e_t|}$. Normalization not only helps to scale dot product into $[-1, 1]$ between nodes but also make it possible use cosine similarity to measure relationship among nodes in ANNS system for retrieval.

Node Encoder. In this part, we introduce three modules to encode nodes from multiple views. To describe the module clear, we suppose the input graph as $\hat{\mathcal{G}} = \{(\hat{q}^i, \hat{s}^i)\}_{i=1}^{\hat{N}}$ with an adjacency matrix $\mathcal{A} \in \mathbb{R}^{\hat{N} \times \hat{N}}$ where $\hat{N}$ is the node number of the sub-graph.

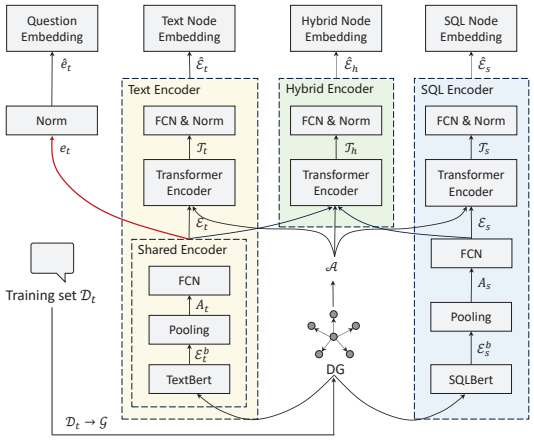

Figure 5: The architecture of DGE.

Text Encoder. The objective of this module is to encode nodes focusing more on text information. We use the NL question embeddings $\mathcal{E}_t = \{e_t^i\}_{i=1}^{\hat{N}}$ of the input graph, generated by the shared encoder, as the features of nodes. To model relationship among nodes in DG, we adopt the self-attention mechanism from the Transformer encoder, but apply an adjacency matrix mask to ensure that each node only attends to its directly connected neighbors in the graph. A 3-layer transformer encoder with GeLU is used to capture node relationship.

$$\mathcal{T}_t = \text{Transformer_Encoder}(\mathcal{E}_t, \mathcal{A}) \tag{3}$$

A three-layer FCN is then applied to introduce non-linearity to the embeddings, which are subsequently normalized to obtain the text-based node embeddings $\hat{\mathcal{E}}_t = \{\hat{e}_t^i\}_{i=1}^{\hat{N}}$.

SQL Encoder. This module aims to encode nodes with a particular focus on SQL queries. The architecture of the SQL encoder is analogous to that of the text encoder. Specifically, we first employ Bert (all-MiniLM-L6-v2) to encode the SQL queries $\{\hat{s}^i\}_{i=1}^{\hat{N}}$ of the graph $\hat{\mathcal{G}}$, where this BERT model is referred to as SQLBert. We then apply average pooling to the token embeddings from the last three layers of SQLBert for each SQL query, resulting in three average-pooled embeddings per query, denoted as $\mathcal{A}_s = \{(a_s^{i,1}, a_s^{i,2}, a_s^{i,3})\}_{i=1}^{\hat{N}}$. Next, a 2-layer FCN maps these average-pooled

vectors to d_t-dimensional embeddings $\mathcal{E}_s = \{e_s^i\}_{i=1}^{\hat{N}}$. Subsequently, a three-layer Transformer encoder with GeLU activation is utilized to model relationships between nodes, producing $\mathcal{T}_s$. Finally, a three-layer FCN is applied to introduce further non-linearity, and the resulting embeddings are normalized to obtain the SQL-based node representations, denoted as $\hat{\mathcal{E}}_s = \{\hat{e}_s^i\}_{i=1}^{\hat{N}}$.

Hybrid Encoder. This module aims to jointly leverage textual and SQL-derived features for node encoding. We first concatenate the embeddings of NL questions ($\mathcal{E}_t$) and SQL queries ($\mathcal{E}_s$). The concatenated representations are then passed through a 3-layer Transformer encoder with GeLU activation to capture node relationships, resulting in $\mathcal{T}_h$. Subsequently, a 3-layer FCN is applied to introduce non-linearity, and the final node embeddings are normalized to obtain $\hat{\mathcal{E}}_h = \{\hat{e}_h^i\}_{i=1}^{\hat{N}}$.

Sub-graph Sampling and Interaction Enhancement. We perform sub-graph sampling from DG to enhance training efficiency of DGE. First, the maximum number of nodes per sub-graph is set to 10. Second, seed nodes are randomly selected from DG. Third, we select the 1-hop neighbors of these seed nodes, with the number of such neighbors limited to half of the maximum sub-graph size. Finally, we iteratively include 2-hop, 3-hop, and further neighboring nodes to fill the remaining slots, until the sub-graph reaches the predefined maximum size or all neighboring nodes have been included. DGE adopts a dual-encoder architecture, which inherently suffers from weak interaction between the two encoders. To address this limitation, DGE uses a simple interaction module that enables information exchange between dual encoders during training, like (Jiang, 2023). Specifically, DGE designs an auxiliary task to predict whether pairs of nodes are linked, facilitating information sharing between the text encoder and the SQL encoder. DGE selects $\hat{\mathcal{E}}_t$ for certain nodes and $\hat{\mathcal{E}}_s$ for others, and uses these representations to enable interaction between the two encoders. To accomplish this, DGE employs an FCN with a sigmoid activation function, allowing the dual encoders to interact as follows:

$$\hat{\mathcal{Y}}_I' = \text{sigmoid}(\text{FCN}(\text{concat}(\hat{\mathcal{E}}_t, \hat{\mathcal{E}}_s))) \tag{4}$$

During training, text encoder and SQL encoder are able to transfer information to each other based on the auxiliary task.

Optimization. We adopt link prediction task, example retrieval task and interaction enhancement task to jointly optimize parameters of DGE. It is noted that we get a new training set $\hat{\mathcal{D}}_t = \{\hat{q}_i, \hat{s}_i\}_{i=1}^{\hat{N}_t}$ after SQL decomposition. Link prediction aims to judge whether two given nodes are linked. Denote the links between any two nodes in the new training set as $\hat{\mathcal{Y}}_L = \{\hat{y}_{i,j}^L\}_{i,j=1}^{\hat{N}_t}$. If the i-th node and the j-th node are linked, $y_{i,j}$ is equal to 1, otherwise it is equal to 0. Denote the link prediction result of DGE as $\hat{\mathcal{Y}}_L' = \{\hat{y}_{i,j}^{L\prime}\}_{i,j=1}^{\hat{N}_t}$, where $\hat{y}_{i,j}^{L\prime}$ is the dot product between the embeddings of the i-th and the j-th nodes. Example retrieval task focuses on judging whether a specific SQL query correctly answers a given question. The ground truth labels and predictions for this task are denoted as $\hat{\mathcal{Y}}_E = \{\hat{y}_{i,j}\}_{i,j=1}^{\hat{N}_t}$ and $\hat{\mathcal{Y}}_E' = \{\hat{y}_{i,j}^{E\prime}\}_{i,j=1}^{\hat{N}_t}$, where $\hat{y}_{i,j}^{E\prime}$ is the dot product between the question embedding of the i-th ndoe and the node embedding of the j-th node. Interaction enhancement task aims to predict the link relationship between nodes using their low-level embeddings. We denote the ground truth and the predictions for this task as $\hat{\mathcal{Y}}_I = \{\hat{y}_{i,j}\}_{i,j=1}^{\hat{N}_t}$ and $\hat{\mathcal{Y}}_I' = \{\hat{y}_{i,j}^{I\prime}\}_{i,j=1}^{\hat{N}_t}$, respectively. The loss function of DGE has three similar parts. For brevity, we take the loss function of text encoding $\mathcal{L}_T$ as an example.

$$\mathcal{L}_T = \alpha\mathcal{L}_L(\hat{\mathcal{Y}}_L', \hat{\mathcal{Y}}_L|\hat{\mathcal{D}}_t) + \beta\mathcal{L}_E(\hat{\mathcal{Y}}_E', \hat{\mathcal{Y}}_E|\hat{\mathcal{D}}_t) + (1-\alpha-\beta)\mathcal{L}_I(\hat{\mathcal{Y}}_I', \hat{\mathcal{Y}}_I|\hat{\mathcal{D}}_t) \tag{5}$$

where $\hat{y}_{ij}^{L\prime} = \hat{e}_t^{i\top}\hat{e}_t^j$ and $\hat{y}_{ij}^{E\prime} = e_q^{i\top}\hat{e}_t^j$. α and β are hyper-parameters, they are set to 0.1 and 0.8 in our experiment. All loss terms ($\mathcal{L}_L$, $\mathcal{L}_E$ and $\mathcal{L}_I$) are formulated using the Noise Contrastive Estimation (NCE) loss.

$$\mathcal{L}_{NCE} = -\log\left(\frac{\exp(\hat{y}'^{+}/\tau)}{\sum_{i=1}^{k}\exp(\hat{y}_i'^{-}/\tau) + \exp(\hat{y}'^{+}/\tau)}\right) \tag{6}$$

where $\hat{y}'^{+}$ is the predicted value of the positive sample, $\hat{y}_i'^{-}$ is the predicted value of the i-th negative sample, k is the number of negative samples, and τ is the temperature coefficient. In our experiment,

Table 1: Performance Analysis on BIRD-Dev dataset. Performances of baselines are referenced from their original papers. The methods marked in red in the table outperform our model. Cost* is based on Qwen3-Coder-30B-A3B, as their original papers do not report the cost. Direct comparison is challenging as each method uses its own optimal LLM and settings.

Method	LLM	ICL	EX (%)				Tokens (K)	Cost ($)
			Easy	Moderate	Challenging	All		
o3-mini	o3-mini	✓	-	-	-	59.52	3.8	-
GPT-4o	GPT-4o	✓	63.35	44.18	45.52	55.87	-	-
Gemini-2.5-pro	Gemini-2.5-pro	✓	-	-	-	66.82	4.7	0.047
Reasoning-SQL	Qwen2.5-Coder-14B	✗	-	-	-	72.29	-	-
OmniSQL	Qwen2.5-Coder-32B	✗	-	-	-	67.0	-	-
CSC-SQL	XiYanSQL-32B	✗	-	-	-	71.3	-	-
XiYan-SQL	GPT-4o	✗	-	-	-	73.34	6.9	-
CHASE-SQL	Gemini-1.5-pro	✗	-	-	-	73.01	208.32	0.424
DIN-SQL	GPT-4	✓	-	-	-	50.72	-	-
DAIL-SQL	GPT-4	✓	63.0	45.6	43.1	55.9	-	-
MAC-SQL	GPT-4	✓	65.73	52.69	40.28	59.39	-	-
MCS-SQL	GPT-4	✓	70.4	53.1	51.4	63.4	-	-
RSL-SQL	GPT-4o	✓	74.4	57.1	53.8	67.21	7.5	-
CHESS	Gemini-1.5-pro	✓	73.48	62.99	59.31	68.31	14.5	0.21*
OpenSearch-SQL	GPT-4o	✓	-	-	-	69.30	26.76	-
AID-SQL	GPT-4o	✓	-	-	-	87.1	-	-
Alpha-SQL	Qwen2.5-Coder-32B	✓	74.5	64.0	57.2	69.7	210.24	0.225*
DeepEye-SQL	Qwen3-Coder-30B	✓	-	-	-	73.5	46.37	0.063
Derivation-SQL	**Gemini-2.5-flash**	✓	**72.00**	**59.27**	**53.79**	**65.78**	**14.99**	**0.007**
Derivation-SQL	**Gemini-2.5-pro**	✓	**76.22**	**64.44**	**65.52**	**71.64**	**14.81**	**0.027**
Derivation-SQL	**Qwen3-Coder-30B**	✓	**69.84**	**56.9**	**48.28**	**63.89**	**12.75**	**0.006**
Derivation-SQL	**GPT-4o**	✓	**77.08**	**66.16**	**64.14**	**72.56**	**11.38**	**0.031**

τ is set to 0.1 and k is set to 5. That means, we randomly select 5 pair nodes as negative samples from the sub-graph, these two nodes within a pair are not linked. Similarly, we obtain loss functions of SQL encoding and Hybrid encoding $\mathcal{L}_S$ and $\mathcal{L}_H$. Therefore, the final loss function of DGE $\mathcal{L}_F$ is below.

$$\mathcal{L}_F = (\mathcal{L}_T + \mathcal{L}_S + \mathcal{L}_H)/3 \tag{7}$$

We use the optimizer Adam (Kingma & Ba, 2015) to minimize the loss over all data.

5 Experiment

5.1 Experimental Setup

Datasets and Metrics. We conduct experiments to evaluate the effectiveness of Derivation-SQL on two widely adopted Text-to-SQL benchmarks: BIRD (Li et al., 2023a) and Spider (Yu et al., 2018). Considering cost and efficiency, we follow previous work CHESS (Talaei et al., 2024) and RSL-SQL (Cao et al., 2024) to perform the ablation analysis and the example number analysis by using a Subsampled Development Set (SDS). Following previous research work, we adopt EXecution accuracy (EX) to evaluate the performance of the proposed method. Besides, token, cost, and running time are utilized to evaluate the efficiency of Derivation-SQL and baseline approaches. For additional experimental details, please refer to Appendix. E.1 to E.4.

Baselines. Derivation-SQL is an ICL-based method, so we mainly compare it with ICL-based baselines. We compare Derivation-SQL with DIN-SQL (Pourreza & Rafiei, 2023), DAIL-SQL (Gao et al., 2024), MAC-SQL (Wang et al., 2025), MCS-SQL (Lee et al., 2025), RSL-SQL (Cao et al., 2024), OpenSearch-SQL (Xie et al., 2025), CHESS (Talaei et al., 2024), Alpha-SQL (Li et al., 2025b) and DeepEye-SQL (Li et al., 2025a). We compare Derivation-SQL with Reasoning-SQL (Pourreza et al., 2025b), OmniSQL (Li et al., 2025c), CSC-SQL (Sheng & Xu, 2025), XiYan-SQL (Liu et al., 2025b), and CHASE-SQL (Pourreza et al., 2025a).

5.2 Performance Analysis

Overview. Table 1 illustrates performances of our method and baselines on BIRD benchmark. We compare the proposed model with vanilla LLM, ICL-based baselines and FT-based baselines sequentially according to their characteristics. By analyzing the performance, we obtain several insights

below: First, Derivation-SQL outperforms most of ICL-based baselines in terms of performance. Although Derivation-SQL with GPT-4o is only marginally weaker than DeepEye-SQL, it achieves this performance using just 50% of the tokens required by DeepEye-SQL. Second, Derivation-SQL outperforms some of FT-based methods and performs closely to CHASE-SQL, which has the best performance. Third, Derivation-SQL with Gemini-2.5-pro is better at solving difficult questions. The EX of Derivation-SQL with Gemini-2.5-pro is much larger than that of any other baseline. Finally, Derivation-SQL is the most cost-effective among all compared methods. It with GPT-4o costs only $0.031 per question, which is half of the cost of the second-cheapest model. Table 2 illustrates performances of our method and baselines on Spider benchmark. After analyzing the experimental results, we obtain: First, the performance of baselines is quite close. Their performances are around 87%. Secondly, Derivation-SQL with Gemini-2.5-pro performs well compared with ICL-based baselines. Among ICL-based approaches, Derivation-SQL with Gemini-2.5-pro only slightly weaker than DeepEye-SQL and MCS-SQL, but the gap is very small.

DGE retrieval Analysis DGE retrieval is a plug-in component. We evaluate its retrieval performance across different Text-to-SQL methods to verify the effectiveness of DGE. Opensearch-SQL and CHESS (the IR_SS_CG version) are chosen for our analysis. In the original papers, Opensearch-SQL utilizes masked similarity to retrieve examples from the original training set, whereas CHESS assigns fixed examples to each question. To achieve better performance, both approaches include explanations illustrating why the SQL query corresponds to the NL question. In both cases, we replace their original examples with our retrieved examples by using DGE. Fig. 7 presents the comparative performance of Opensearch-SQL and CHESS. Our analysis shows that DGE retrieval not only improves the performance of both methods but also reduces token consumption, as it provides more relevant examples and achieves excellent performance without explanations. These results demonstrate the effectiveness of DGE retrieval in enhancing both the accuracy and efficiency of Text-to-SQL approaches.

Table 2: Performance Analysis on Spider-Test dataset. Performances of baselines are referenced from their original papers.

Method	LLM	ICL	EX (%)
OmniSQL	Qwen2.5-Coder-32B-Instruct	✗	89.8
CHASE-SQL	Gemini-1.5-pro	✗	87.6
XiYan-SQL	GPT-4o	✗	89.7
DIN-SQL	GPT-4	✓	85.3
DAIL-SQL	GPT-4	✓	86.6
MAC-SQL	GPT-4	✓	86.75
MCS-SQL	GPT-4	✓	89.6
CHESS	Gemini-1.5-pro	✓	87.2
OpenSearch-SQL	GPT-4o	✓	87.1
RSL-SQL	GPT-4o	✓	87.9
DeepEye-SQL	Qwen3-Coder-30B-A3B	✓	89.8
Derivation-SQL	**Gemini-2.5-flash**	✓	**86.49**
Derivation-SQL	**Gemini-2.5-pro**	✓	**88.93**
Derivation-SQL	**Qwen3-Coder-30B-A3B**	✓	**81.51**
Derivation-SQL	**GPT-4o**	✓	**88.08**

Error Analysis. We analyze the error factor proportion in the Gemini-2.5-pro results, as shown in Fig. 8. Schema and JOIN errors are the most common limitations for Derivation-SQL, indicating potential areas for further optimization.

Ablation Analysis. To assess the importance of each module, we conduct an ablation study. For cost efficiency, the experiments are performed using Gemini-2.5-flash on the BIRD-SDS and Spider-SDS datasets. Table 6 presents the results. Several key observations can be drawn: First, the three types of retrieval strategies in Derivation-SQL play a crucial role. Removing any of them results in a substantial performance drop, particularly on BIRD-SDS. Second, Derivation-SQL outperforms the case DGE→Bert, showing a performance gap of 2.99% on BIRD-SDS and 2.97% on Spider-SDS. This strongly supports the effectiveness of our approach. Third, models employing few-shot methods achieve better results than the case w/o few-shot, highlighting the essential role of few-shot in Text-to-SQL tasks. To reduce the influence of diversity, we try each strategy three times and report the performance. The results indicate that combining all three strategies (our Derivation-SQL) achieves the best performance. This confirms that the improvements are due to our retrieval

Figure 6: Ablation Analysis on BIRD-SDS and Spider-SDS. * means that it was repeated three times and the selection module was used to select one.

Methods	BIRD EX (%)	Spider EX (%)
Derivation-SQL	71.1	85.48
w/o Text-based retrieval	68.77 (-2.33)	84.49 (-0.99)
w/o SQL-based retrieval	69.10 (-2.00)	84.16 (-1.32)
w/o Hybrid retrieval	69.44 (-1.66)	83.50 (-1.98)
DGE → Bert	68.44 (-2.66)	83.83 (-1.65)
w/o few-shot	68.11 (-2.99)	82.51 (-2.97)
only Text-based retrieval*	69.10 (-2.00)	83.50 (-1.98)
only SQL-based retrieval*	69.44 (-1.66)	84.16 (-1.32)
only Hybrid retrieval*	70.10 (-1.00)	83.17 (-2.31)
DGE → Bert*	69.77 (-1.33)	84.16 (-1.32)

Table 3: Efficiency Analysis on BIRD. The result is for each question. The cost of CHASE-SQL is referenced from (Pourreza et al., 2025b). * means that the module runs offline.

Methods	LLM	Time Latency (s)	Input Tokens (K)	Output Tokens (K)	Cost ($)	EX (%)
CHESS	Qwen3-30B-A3B	284.4	327.02	27.83	0.210	67.9
Opensearch-SQL	GPT-4o	60.3	22.03	4.73	0.102	69.3
Alpha-SQL	Qwen3-Coder-30B-A3B	377.1	138.03	72.21	0.225	71.2
DeepEye-SQL	Qwen3-Coder-30B-A3B	17.8	23.21	23.16	0.063	73.5
CHASE-SQL	Gemini-1.5-pro	-	164.76	43.57	0.424	74.46
Derivation-SQL	**GPT-4o**	**7.5**	**11.05**	**0.33**	**0.031**	**72.56**
Encoding	-	2.4×10^{-3}	-	-	-	-
Retrieval	-	0.39	-	-	-	-
Information Extraction	GPT-4o	1.0	0.07	0.36	-	-
Schema Linking	GPT-4o	1.3	3.62	0.06	-	-
SQL Generation	GPT-4o	1.3	5.55	0.17	-	-
Correction	GPT-4o	0.4	0.59	0.04	-	-
Selection	GPT-4o	0.9	0.62	0.01	-	-
Decomposition*	-	1.1×10^{-3}	-	-	-	-
DGE Training*	-	2400	-	-	-	-
Question Generation*	Gemini-2.5-flash	1.6	1.88	0.09	0.002	-

mechanism. In summary, the ablation analysis demonstrates the effectiveness and necessity of each component in Derivation-SQL.

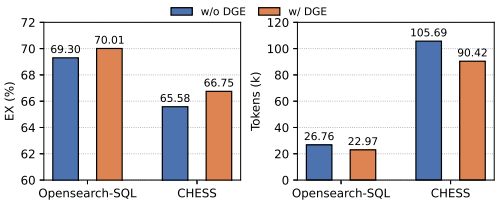

Figure 7: DGE retrieval Analysis.

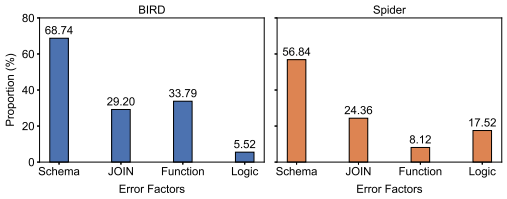

Figure 8: Error Analysis.

Efficiency Analysis. Table 3 compares the efficiency of our proposed method with those of baselines whose performance is similar to ours. We regard the monetary cost per question as an indicator of efficiency, since a lower cost implies that the method achieves its goal with fewer resources. The cost of offline module is not a major concern for Derivation-SQL, as both the model and example updates are infrequent. We compare Derivation-SQL with baselines and draw several insights: First, Derivation-SQL is the fastest method. Baselines spend at least more than 1.37x time on each NL question. Second, Derivation-SQL is the cheapest method. Derivation-SQL saves about 50% money cost compared to baselines. Third, our method exhibits significantly lower token consumption than most baselines. Derivation-SQL consumes the fewest for both input tokens and output tokens. We analyze the latency and token usage of each module in Derivation-SQL. Several key observations can be made: First, schema linking and SQL generation account for the majority of token usage. This is primarily because schema requirements involve providing information for 40 columns as input to the LLM during SQL generation, and the model generates SQL queries three times, each using different retrieved examples. Second, the Correction module consumes the fewest tokens. Since the Correction module is invoked only a limited number of times, its average token consumption is extremely low when aggregated.

6 Conclusion

In this paper, we proposed Derivation-SQL, a cost-effective graph-based framework that explicitly models SQL derivation to retrieve relevant examples for Text-to-SQL. Unlike text-similarity-based methods, Derivation-SQLconstructs a SQL derivation graph and leverages a derivation-aware embedding network with efficient Node and Schema ANNS to capture deep structural correlations. Extensive experiments on benchmark datasets demonstrate that Derivation-SQL attains a Pareto-optimal balance between accuracy and cost, maintaining strong competitive performance at substantially lower computational overhead. Ablation studies further validate the contribution of each component.

AI-GENERATED CONTENT ACKNOWLEDGEMENT

The authors used GPT 4.1 to improve the language clarity and readability of this paper. Pictures and code are not generated by AI tools.

REFERENCES

Qifeng Cai, Hao Liang, Chang Xu, Tao Xie, Wentao Zhang, and Bin Cui. Text2sql-flow: A robust sql-aware data augmentation framework for text-to-sql. *CoRR*, abs/2511.10192, 2025. doi: 10.48550/ARXIV.2511.10192. URL https://doi.org/10.48550/arXiv.2511.10192.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. RSL-SQL: robust schema linking in text-to-sql generation. *CoRR*, abs/2411.00073, 2024.

Jaime G. Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pp. 335–336. ACM, 1998.

Yaxun Dai, Haiqin Yang, Mou Hao, and Pingfu Chao. Parsql: Enhancing text-to-sql through sql parsing and reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 661–681, 2025.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. C3: zero-shot text-to-sql with chatgpt. *CoRR*, abs/2307.07306, 2023.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. PASTA: table-operations aware fact verification via sentence-table cloze pre-training. In *EMNLP*, pp. 4971–4983. Association for Computational Linguistics, 2022.

Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. Few-shot text-to-sql translation using structure and content prompt learning. *Proc. ACM Manag. Data*, 1(2): 147:1–147:28, 2023.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Thanh Dat Hoang, Thanh Trung Huynh, Matthias Weidlich, Thanh Tam Nguyen, Tong Chen, Hongzhi Yin, and Quoc Viet Hung Nguyen. FINER-SQL: boosting small language models for text-to-sql. *CoRR*, abs/2605.03465, 2026. doi: 10.48550/ARXIV.2605.03465. URL https://doi.org/10.48550/arXiv.2605.03465.

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

Junxiang Jiang. TADI: topic-aware attention and powerful dual-encoder interaction for recall in news recommendation. In *Findings of EMNLP*, pp. 15647–15658. Association for Computational Linguistics, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pp. 337–353. Association for Computational Linguistics, 2025.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. Deepeye-sql: A software-engineering-inspired text-to-sql framework. *CoRR*, abs/2510.17586, 2025a.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *CoRR*, abs/2502.17248, 2025b.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127, 2024a.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709, 2025c.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *NeurIPS*, 2023a.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*, 2023, 2023b.

Xiuwen Li, Qifeng Cai, Yang Shu, Chenjuan Guo, and Bin Yang. AID-SQL: adaptive in-context learning of text-to-sql with difficulty-aware instruction and retrieval-augmented generation. In *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*, pp. 3945–3957. IEEE, 2025d.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. PET-SQL: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *CoRR*, abs/2403.09732, 2024b.

Weibin Liao, Xin Gao, Tianyu Jia, Rihong Qiu, Yifan Zhu, Yang Lin, Xinyu Ma, Junfeng Zhao, and Yasha Wang. Learnat: Learning nl2sql with ast-guided task decomposition for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 87420–87452, 2026.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Trans. Knowl. Data Eng.*, 37(10):5735–5754, 2025a.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. Xiyan-sql: A novel multi-generator framework for text-to-sql. *CoRR*, abs/2507.04701, 2025b.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*, 2023.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. Reasoning-sql: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*, abs/2503.23157, 2025b.

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, Kaustubh Vyas, Yuanyi Ji, and Jeff Z. Pan. Improving retrieval-augmented text-to-sql with ast-based ranking and schema pruning. In *EMNLP*, pp. 7865–7879. Association for Computational Linguistics, 2024.

Lei Sheng and Shuai-Shuai Xu. CSC-SQL: corrective self-consistency in text-to-sql via reinforcement learning. *CoRR*, abs/2505.13271, 2025.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2):54:1–54:37, 2026.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, 2014.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain of thought style prompting for text-to-sql. In *EMNLP*, pp. 5376–5393. Association for Computational Linguistics, 2023.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755, 2024.

Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K Kummerfeld, and Tianyi Zhang. Interactive text-to-sql generation via editable step-by-step explanations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 16149–16166, 2023.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pp. 540–557. Association for Computational Linguistics, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Proc. ACM Manag. Data*, 3(3):194:1–194:24, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, pp. 3911–3921. Association for Computational Linguistics, 2018.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir R. Radev. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4511–4523. Association for Computational Linguistics, 2019.

A DGE-Enhanced Schema Linking and SQL Generation

DGE encodes both natural language (NL) questions and examples by leveraging the derivation relationships among SQL queries, enabling us to retrieve more relevant examples for each question. However, a gap remains between the retrieved examples and the final SQL queries we generate. A

key challenge stands before us: how to effectively utilize these retrieved examples to assist in SQL query generation. In this section, we introduce the remaining module of Derivation-SQL, which integrates the retrieved examples with existing information, such as the database schema, to generate SQL queries.

Schema Linking. Its goal is to identify the relevant databases, tables and columns for the given question. Afterwards, the selected schema could be integrated with the prompts $\mathbf{P}$. For clarity, we categorize smaller modules within schema linking into two parts: offline and online. Offline modules could be completed before the user types the question. They contain the following modules. First, database schema description. An LLM is used to describe the meanings of databases, tables, and columns based on the database schema and incomplete comments. Second, database schema encoding and storage. By using BERT (all-MiniLM-L6-v2), we encode the meanings of databases, tables, and columns; the names of databases, tables, and columns; and data from the database. By doing so, we store the embeddings into ANNS. The online modules should immediately run if the user types their question. They include many modules. First, information extraction of inputs. We use an LLM to extract keywords and key phrases from the question (and the hint if it is provided). Second, relevant database schema retrieval. This module retrieves columns that are used in the SQL query. Encoding the extracted information with BERT (all-MiniLM-L6-v2) at first. Then, retrieving related database schema from the ANNS. We follow DIN-SQL (Pourreza & Rafiei, 2023) to use the given db_id to filter the database schema at the beginning. Then, we separately retrieve tables and columns, and score the column as $\hat{s}_c$ considering views of the database, table, and column together.

$$\hat{s}_c = 0.5 \times s_t + 0.5 \times s_c \tag{8}$$

s_t and s_c are the cosine similarities between the question and a table or column. Since multiple keywords and key phrases are used for schema retrieval, a single schema element may be associated with multiple similarity scores. We take the maximum value as the final similarity between the question and each schema element. Based on these similarity scores, the top 40 columns are selected in descending order. Finally, an LLM is employed to explicitly select the relevant columns from these 40 candidates.

SQL Generation. The first step of this module is to obtain relevant examples from the training set to guide LLMs in generating SQL queries. The process is as follows: First, each example is encoded into three dense vectors using DGE in the offline stage. These vectors are then stored in an ANNS with a FlatIndex structure. Third, correlated examples are retrieved online: when users input a question, Derivation-SQL immediately retrieves related examples from the ANNS. Finally, retrieved examples are re-ranked. We observed that using highly similar examples can negatively impact performance. To address this, we apply the Maximal Marginal Relevance (MMR) algorithm (Carbonell & Goldstein, 1998) to re-rank the selected examples. Specifically, a larger candidate set (15 examples in our experiments) is first retrieved. The MMR algorithm is then used to re-rank them, and only a subset of examples is finally selected for inclusion. The second step is to translate the NL question into an SQL query. In this step, achieving this goal primarily relies on effective prompting. We follow (Pourreza et al., 2025a; Li et al., 2025b), the prompt incorporates a divide-and-conquer strategy, which guides the LLM to decompose the NL question into sub-questions and solve each sub-question individually. The final answer to the original question is then constructed from the answers to these sub-questions. Moreover, we employ an LLM to generate three SQL queries based on the three types of examples retrieved by DGE.

Post-processing. , Post Processing consists of two key modules: Correction and Selection. The objective of the correction module is to repair incorrect SQL queries generated by the model, utilizing LLMs to produce correct, executable SQL queries. When the database identifies an erroneous query, the failed SQL queries, relevant database schema, and the error message returned by the database are provided as input to the LLM. The LLM then attempts to return a corrected SQL query. This correction process is repeated up to three times per query. If the query cannot be fixed after three attempts, the original SQL query is returned. The objective of the selection module is to identify the most likely correct SQL query for the given NL question from the three candidates generated in the SQL generation module. The NL question, database schema, and all three SQL queries are provided in the prompt, and the LLM is tasked with selecting the most appropriate query.

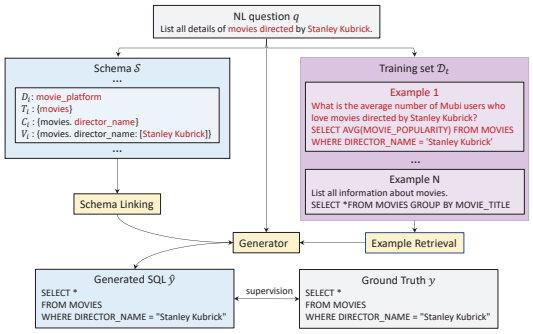

Figure 9: An example of Notations. The red words are relevant content.

B Problem Formulation

Database Schema. Following the M-schema construction (Liu et al., 2025b), the schema $\mathcal{S}$ comprises N_s databases. For the i-th database, the components of $(D_i, \mathcal{T}_i, \mathcal{C}_i, \mathcal{V}_i)$ are defined as follows: (1) **Database Information** D_i: the database name and all foreign keys. (2) **Table Information** $\mathcal{T}_i$: the table name and the columns belonging to this table. (3) **Column Information** $\mathcal{C}_i$: the column name, data type, whether it is a primary key, and a textual description. (4) **Value Information** $\mathcal{V}_i$: example values of a column sampled from the database. To keep computation manageable, we retain only five distinct values per column.

C SQL Derivation Graph Construction

C.1 Algorithm

Algorithm 1 builds a derivation graph DG by interleaving decomposition with graph construction. The input operation list O_{rev} enumerates all SQL clause types; removing an outermost occurrence is deterministic. The algorithm first initialises the graph (V, E), the set $\hat{D}_t$ of processed pairs, and a queue Q with all original pairs (lines 1–3). In the main loop (lines 5–17), each pair (q, s) is processed exactly once: if it has been handled, it is skipped (lines 7–8); otherwise it is recorded in $\hat{D}_t$ and a fresh node v is created (line 9). Node identity relies on its SQL query s, and each SQL is associated with a question. For every operation $op \in O_{\text{rev}}$, the algorithm strips the outermost occurrence from s (line 12). When a valid sub-query s_{sub} is obtained, the corresponding node u (labeled $(\text{None}, s_{\text{sub}})$) is created if it does not already exist, and enqueued (lines 14–16). A directed edge $\langle u, v \rangle$ is inserted, meaning v was obtained by adding op to u (line 17). Since edges always point from a simpler query to a more complex one, DG is a DAG. The queue ensures breadth-first exploration until no new sub-queries appear. Question generation (lines 18–32) first iteratively processes nodes whose majority of neighbours already have a question, enriching the LLM context (lines 19–29). The loop allows previously ineligible nodes to become eligible as neighbour questions increase. Finally, any remaining nodes are directly processed by the LLM without neighbour context (lines 30–32), guaranteeing that every node receives a question. The fully annotated graph DG is returned (line 33).

C.2 Evaluation.

We evaluate the syntax and semantics of decomposed SQL queries using database management system (DBMS). On the BIRD dataset excluding those queries which could not executed by DBMS, 96.75% of the decomposed SQL queries can be successfully executed except the . The errors mainly stem from the diversity and complexity of certain SQL queries, such as deeply nested structures and rare operators. By analyzing the performance on the downstream task (Text-to-SQL), we confirm that the decomposed SQL queries are beneficial for improving Text-to-SQL results.

C.3 Derivation Graph Construction Strategy

DG construction employs a pragmatic, heuristic-based decomposition algorithm. This approach is designed to aggressively decompose queries by recursively stripping clauses (SELECT, FROM,

Table 4: Analysis of SQL Decomposition.

Methods	Success Rate (%)	Tokens (K)
CoT	68.11	16,735
LEARNAT	80	145,977
Derivation-SQL	99.6	30,629

WHERE, etc.) and extracting sub-queries. The primary goal is to build a densely connected graph that captures a rich set of relationships between SQL queries, a crucial element for the success of our graph embedding and retrieval process.

A known limitation of this heuristic approach arises when encountering SQL structures with strong semantic coupling, such as an aggregate function in a SELECT list and its corresponding GROUP BY clause, or certain advanced window functions. A sequential, clause-by-clause removal can result in syntactically invalid fragments (e.g., SELECT id, COUNT(id) FROM a). These non-executable fragments constitute a small fraction of the generated nodes (3.25%).

Our strategy for handling these fragments is a deliberate, two-phase design that balances graph connectivity with example quality:

Phase 1: Prioritizing Connectivity on Graph Construction. We retain these non-executable fragments within the graph. Instead of being discarded, they serve as essential structural placeholders or ”bridges”. This ensures the graph remains fully connected, allowing our graph embedding model to learn from the complete derivational topology. Without these bridges, the graph would splinter into disconnected components, impoverishing the learned structural representations.

Phase 2: Prioritizing Quality on few-shot Retrieval. We implement a strict validity filter during the retrieval process. When selecting few-shot examples to include in the prompt for the LLM, we exclusively consider nodes that represent valid, executable SQL queries. The non-executable placeholder nodes are explicitly excluded from the few-shot example candidates.

This two-phase strategy allows us to leverage the strengths of graph connectivity and example quality: a comprehensive and connected graph for robust structural learning, and a high-quality, reliable set of examples to guide the LLM. It effectively prevents the context of the LLM from being contaminated by incorrect syntax, while still benefiting from the rich relational information embedded in the complete graph structure. Future work could involve enhancing our decomposition algorithm with deeper semantic analysis to reduce the creation of these placeholder nodes, but our current strategy ensures both robustness and high performance.

C.4 Comparison

We compare the decomposition methods of CoT, Derivation-SQL and LEARNAT (Liao et al., 2026), as shown in Table 4. The success rate refers to that the cumulative execution of these sub-SQL queries is equivalent to the original SQL query. Derivation-SQL achieves superior performance. The token consumption for Derivation-SQL is due to NL question generation, which is lower than that of the other methods.

C.5 Complexity Analysis.

Let $N = |V|$ be the total number of nodes in graph DG, and L the average length of a SQL string. The operation list O_{rev} has constant size $c = 9$, and each outermost removal takes $O(L)$ time. The main loop processes each unique pair exactly once (enqueued at most once per creation, and skipped if already in $\hat{D}_t$). For each processed pair, c removal attempts are made, and node existence checks / edge insertions take $O(1)$ amortized time. Hence, decomposition and graph construction cost $O(N \cdot c \cdot L) = O(NL)$. Question generation calls the LLM exactly once per node (either in the iterative majority-neighbour phase or the final sweep). With T_{LLM} the average inference time, this phase costs $O(N \cdot T_{\text{LLM}})$. The overall time complexity is $O(NL + N \cdot T_{\text{LLM}})$, linear in N under practical bounds on L and T_{LLM}. Space complexity is $O(N + |E|) = O(N)$, since $|E| \leq cN$ and auxiliary structures store at most N elements.

Algorithm 1: SQL Derivation Graph Construction

Input: Training set $\mathcal{D}_t = \{(q_t^i, s_t^i)\}_{i=1:N_t}$, operation list
$\mathcal{O}_{\text{rev}} = (\text{UNION}, \text{LIMIT}, \text{ORDERBY}, \text{HAVING}, \text{SELECT}, \text{GROUPBY}, \text{WHERE}, \text{JOIN}, \text{FROM})$
Output: Graph $DG = (V, E)$, where each node $v \in V$ corresponds to a question-SQL pair (q, s)

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$;
$\hat{D}_t \leftarrow \emptyset$, $Q \leftarrow$ empty queue;
for *each* $(q, s) \in D_t$ **do**
  Q.push((q, s));
while $Q \neq \emptyset$ **do**
  $(q, s) \leftarrow Q$.pop();
  if $(q, s) \in \hat{D}_t$ **then**
    continue
  $\hat{D}_t \leftarrow \hat{D}_t \cup \{(q, s)\}$;
  Create a node v labelled (q, s); $V \leftarrow V \cup \{v\}$ **for** *each op* $\in O_{rev}$ **do**
    $s_{\text{sub}} \leftarrow$ remove the outermost *op* from s (if present; otherwise $\perp$);
    if $s_{sub} \neq \perp$ **then**
      Let u be the node labelled (None, s_{sub});
      if *no node with label* (*None*, s_{sub}) *exists in* V **then**
        $V \leftarrow V \cup \{u\}$;
        Q.push((None, s_{sub}));
      $E \leftarrow E \cup \{< u, v >\}$; `/* edge from sub-query to complex query */`
$V_{\text{noQ}} \leftarrow \{v \in V \mid \text{question of } v \text{ is None}\}$;
$changed \leftarrow$ True;
while $V_{noQ} \neq \emptyset$ ***and*** $changed$ **do**
  $changed \leftarrow$ False;
  for *each* $v \in V_{noQ}$ *with SQL* s_v **do**
    $N(v) \leftarrow$ neighbours of v in DG;
    $K(v) \leftarrow \{u \in N(v) \mid \text{question of } u \neq \text{None}\}$;
    if $2 \cdot |K(v)| > |N(v)|$ **then**
      $q_v \leftarrow \text{LLM}(\text{schema of } v, \{(q_u, s_u)\}_{u \in K(v)}, s_v)$;
      Replace the question label of v with q_v;
      $V_{\text{noQ}} \leftarrow V_{\text{noQ}} \setminus \{v\}$;
      $changed \leftarrow$ True;
for *each* $v \in V_{noQ}$ *with SQL* s_v **do**
  $q_v \leftarrow \text{LLM}(\text{schema of } v, \emptyset, s_v)$;
  Replace the question label of v with q_v;
return $DG = (V, E)$;

D Derivation-SQL Embedding

D.1 Complexity Analysis

We analyze the parameter count and time complexity of DGE to demonstrate its efficiency.

Analysis of Model Parameters. Space complexity of DGE is related to parameters of the model. Therefore, we analyze the parameters of DGE. For each Bert part, parameters could be categorized into the embedding part and the transformer layer part. Parameters of the embedding part is $V \times H$ where H is the hidden dimension, V is the vocabulary size. Parameters of each transformer layer is $12 \times H^2$. Therefore, parameters of all-MiniLM-L6-v2 is $30522 \times 384 + 6 \times 12 \times 382^2 = 22M$. GAT is a graph version of transformer, therefore, parameters of GAT is $3 \times 3 \times 12 \times 384^2 = 16M$. In summary, parameters of DGE is $2 \times 22M + 16M = 60M$. Compared with the parameters of LLM, the parameters of DGE is negligible.

Time complexity. The dominant parts of Derivation-SQL are the two Berts and the GAT where these two Berts could be paralleled. Besides, we adopt many FCN. After dropping lower order terms and constant factors, the time complexity of the Bert part, the GAT part and the FCN part are $O(N_b^2 d_b)$,

Table 5: Dataset Statistic.

Dataset	Samples	Train	Dev	Test	DB	Table
BIRD (Li et al., 2023a)	12,751	9,428	1,534	1,789	95	693
Spider (Yu et al., 2018)	10,181	8,659	1,034	2,147	200	1,020

Nd_g^2 and d_f^2. N_b is the maximum input length (bounded). d_b is the Bert hidden dimension. d_g is the GAT hidden dimension. d_f is the FCN hidden dimension. N is the node number. In summary, DGE's time complexity is $O(N_b^2 d_b + Nd_g^2 + d_f^2)$. The time complexity of graph construction is $O(N + M)$ where M is the edge number. Thus, DGE and graph construction scales linearly in graph size.

E Experiment

E.1 Dataset and Metrics

We conduct experiments to evaluate the effectiveness of Derivation-SQL on two widely adopted Text-to-SQL benchmarks: BIRD (Li et al., 2023a) and Spider (Yu et al., 2018). BIRD is a large-scale real-world database benchmark containing 9,428 training samples, 1,534 development samples, 1,789 test samples, and 95 databases. Notably, databases from the training set and the development set are not overlapped. That means the performance on this benchmark reflects the model generalization. Spider is an easier database benchmark containing 8,659 training samples, 1,034 development samples, 2,147 test samples, and 200 databases. Considering cost and efficiency, we follow previous work CHESS (Talaei et al., 2024) and RSL-SQL (Cao et al., 2024) to perform the ablation analysis and the example number analysis by using a Subsampled Development Set (SDS). BIRD-SDS subsamples 20% of each database in the development set of BIRD, and the sample number is 301. Spider-SDS subsamples 30% of each database in the development set of Spider and it has 303 samples. Following previous research work, we adopt EXecution accuracy (EX) to evaluate the performance of the proposed method. EX is to evaluate model performance by comparing whether the execution results of the ground-truth and the generated SQL queries are completely identical. Besides, token, cost and running time are utilized to evaluate the efficiency of Derivation-SQL and baseline approaches.

E.2 baselines

Derivation-SQL is an ICL-based method, so we mainly compare it with ICL-based baselines. ICL-based baselines address the Text-to-SQL task by optimizing prompts. We compare Derivation-SQL with DIN-SQL (Pourreza & Rafiei, 2023), DAIL-SQL (Gao et al., 2024), MAC-SQL (Wang et al., 2025), MCS-SQL (Lee et al., 2025), RSL-SQL (Cao et al., 2024), OpenSearch-SQL (Xie et al., 2025), CHESS (Talaei et al., 2024), Alpha-SQL (Li et al., 2025b) and DeepEye-SQL (Li et al., 2025a). Besides, we compare Derivation-SQL with many vanilla LLMs and FT-based methods. The objective of taking vanilla LLMs into comparison is to demonstrate the effectiveness of prompting. Especially, vanilla LLMs recently achieved significant advancement, and thus the comparison becomes more necessary. The comparable vanilla LLMs are o3-mini, GPT-4o and Gemini-2.5-pro. The FT-based approaches solve Text-to-SQL task by fine-tuning LLMs. We compare Derivation-SQL with Reasoning-SQL (Pourreza et al., 2025b), OmniSQL (Li et al., 2025c), CSC-SQL (Sheng & Xu, 2025), XiYan-SQL (Liu et al., 2025b), and CHASE-SQL (Pourreza et al., 2025a). It is important to note that the comparison between Derivation-SQL and FT-based approaches is not entirely fair and primarily serves to showcase the performance of Derivation-SQL. The closer the performance of Derivation-SQL is to that of FT-based methods, the more effective Derivation-SQL proves to be. This is significant because FT-based methods involves training LLMs, which generally yields stronger results but requires substantial computational resources.

Baselines are reported with their originally published LLMs and settings. Costs are a mix of reported figures and estimations (marked with *) from other papers for a holistic view of the performance-cost trade-off. A direct comparison is challenging due to the heterogeneity of setups.

E.3 IMPLEMENTATION

All experiments are performed on a system equipped with an NVIDIA GeForce RTX 5060Ti GPU, an Intel Core i7-14700 CPU (2.10 GHz), and 32 GB RAM. Our DGE, which incorporates both a TextBERT and a SQLBERT component, was trained for 10 epochs. We used the OneCycleLR scheduler with a maximum learning rate of $1e^{-3}$.

E.4 LLM CONFIGURATION

Throughout this paper, we evaluate Derivation-SQL with GPT-4o, Gemini-2.5-pro, Gemini-2.5-flash and Qwen3-Coder-30B-A3B. Since Gemini-2.5-pro and Gemini-2.5-flash are thinking models, a significant proportion of the computational cost is dedicated to the thinking output. To adhere to our goal of achieving excellent performance with minimal resource consumption, we limit the thinking_budget to 128, which substantially reduces token usage. All other hyper-parameters are kept at their default values; for example, the temperature is set to 1.0, top_p to 0.95, and top_k to 64.

Due to the closed-source nature of some LLMs, we are unable to directly analyze their efficiency. Therefore, we evaluate efficiency by measuring the monetary cost consumed per question translation. We use the token and LLM price to calculate the cost. The formulation is given below:

$$cost = input_price \times input_tokens + completion_price \times completion_tokens \tag{9}$$

Notably, we do not consider the influence of KV cache because almost questions are unique.

The official pricing for each LLM is listed as follows: For Gemini-2.5-flash, the cost is $0.30 per 1M input tokens and $2.50 per 1M completion tokens. For Gemini-2.5-pro, the cost is $1.25 per 1M input tokens and $10.00 per 1M completion tokens. For Qwen3-Coder-30B-A3B, the cost is $0.45 per 1M input tokens and $2.25 per 1M completion tokens. It is worth noting that Qwen3-Coder-30B-A3B is open-source; deploying it independently may be considerably cheaper.

E.5 LLM ANALYSIS

Derivation-SQL is LLM-agnostic, allowing for easy replacement of underlying LLMs. To evaluate the performance of our proposed method more comprehensively, we conduct experiments using different LLMs on the BIRD and Spider datasets. Table 1 and Table 2 illustrate the performance of Derivation-SQL with various LLMs. Several observations can be drawn from the figure. First, Derivation-SQL generally achieves better performance when paired with Gemini-2.5-pro and GPT-4o compared to the rest of LLMs, suggesting that complex models may be more suitable for Derivation-SQL. Second, Derivation-SQL with Gemini-2.5-pro yields the high performance on both BIRD and Spider. On Spider, the performance of Derivation-SQL using Gemini-2.5-flash is close to that using Gemini-2.5-pro, indicating that Gemini-2.5-flash can be employed for less challenging questions, which is an effective way to reduce computational costs.

Error Analysis. we analyze the error factor proportion in the Gemini-2.5-pro results, as shown in Fig. 8. There includes four error categories: Schema error, JOIN error, Function error, and Logic error. Schema error means that the schema is used incorrectly in the generated SQL queries. JOIN error means that the generated SQL queries contain redundant or missing JOIN operations. Function error means that the generated SQL queries use wrong function or miss some functions. Logic error indicates that the generated SQL queries cannot be executed successfully. Schema and JOIN errors are the most common limitations for Derivation-SQL, indicating potential areas for further optimization.

E.6 ANALYSIS ON LLM-GENERATED QUESTIONS

Because we use an LLM to generate NL questions for the SQL queries obtained after decomposition, it is necessary to evaluate the quality of these generated questions. We conduct an automated quality assessment on LLM-generated NL questions, as shown in Table 6. We use a schema linking task and LLM-as-a-judge task to assess quality of LLM-generated NL question. Schema linking task is one of the core sub-tasks of Text-to-SQL, and the performance on schema linking could reflect the quality of the generated NL questions on Text-to-SQL. We follow the setting of our Text-to-SQL experiment: retrieving the top 40 relevant candidates. For cost efficiency, we conduct experiment on

Table 6: Generated Question Quality Analysis.

Dataset	Type	Number	Precision (%)	Recall (%)	LLM Rating
BIRD	Human Question	9,428	10.35	90.69	4.28
	Generated Question	15,641	8.64	80.54	4.47
Spider	Human Question	8,659	7.39	95.5	4.76
	Generated Question	3,098	6.43	81.34	4.64

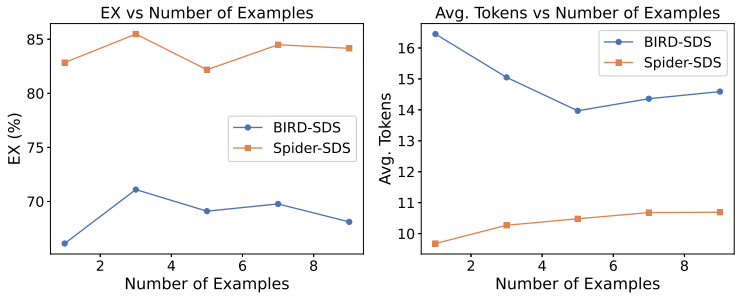

Figure 10: Analysis of Example Number.

300 random samples. After analyzing the results, we find that: the generated questions show slightly lower, but comparable, precision/recall to the human-written questions, confirming high quality. LLM-as-a-judge task is to use a high-performing LLM to evaluate the generated NL questions. Gemini-2.5-pro is adopted as the evaluator, assigning a score from 1 to 5 for each question. A higher score indicates better alignment between the generated question and the corresponding SQL query. As shown in the table, our results confirm that the quality of the generated questions is comparable to that of human-written questions.

E.7 Analysis of The Example Number

Derivation-SQL is highly sensitive to the number of examples used in the prompt. We analyze the impact of the example number on performance across BIRD-SDS and Spider-SDS. Figure 10 illustrates performance trends with varying numbers of examples. On BIRD-SDS, Derivation-SQL achieves optimal performance when the number of examples is set to 3. On Spider-SDS, the best performance is observed when using 3 or 5 examples. Considering that increasing the number of examples in the prompt leads to higher token consumption, we select 3 examples as a balanced trade-off between performance and efficiency. In summary, the optimal number of examples for both BIRD-SDS and Spider-SDS is 3.

E.8 Analysis of Different Retrieval Strategies on Derivation-SQL

We additionally fix the LLM, prompt, k, and downstream pipeline and vary only the retriever. Retriever baselines include random, Semantic-based retrieval, SQL-structure-based retrieval and AST-based retrieval (Shen et al., 2024). Semantic-based retrieval encodes the natural language questions using BERT. SQL-Structure-based retrieval (Gao et al., 2024) first generates a predicted SQL query, then encodes the SQL using BERT, and finally retrieves few-shot example candidates based on the embedding of the predicted SQL. AST-Based retrieval (Shen et al., 2024) also generates a predicted SQL query first, and then measures similarity between the predicted SQL and the candidate examples using the AST structure. The results are summarized in Table 7. DGE performs best in both controlled settings, showing consistent effectiveness across two LLMs.

F Related Work

In this section, we separately review related research work on foundational LLM approaches to Text-to-SQL and few-shot-enhanced strategies.

Table 7: Analysis of Different Retrieval Strategies on Derivation-SQL.

LLMs	Random	Semantic	SQL-structure	AST-based	**DGE**
Gemini-2.5-flash	68.11	70.10	69.44	68.44	**71.10**
GPT-4o	65.45	64.78	68.11	63.79	**73.09**

Table 8: Comparison of few-shot-based Approaches. "Beyond Training set" means that the used examples are not merely the original training set. "wo/ External" means that the method does not adopt external information. "wo/ Synthetic SQL" means that the method does not use LLMs to synthesize additional SQL queries.

Method	Example Source			Retrieval Method	Retrieval Cost	EX
	Beyond Training set	wo/ External	wo/ Synthetic SQL			
ASTRES (Shen et al., 2024)	×	✓	✓	AST Similarity	★★★	★
PET-SQL (Li et al., 2024b)	×	✓	✓	Sentence Transformer	★★★	★
MCS-SQL (Lee et al., 2025)	×	✓	✓	hybrid question similarity	★★★	★
DAIL-SQL (Gao et al., 2024)	×	✓	✓	all-mpnet-base-v2	★★★	★
Opensearch-SQL (Xie et al., 2025)	×	✓	✓	all-mpnet-base-v2	★★★	★★
QDecomp (Tai et al., 2023)	×	✓	✓	Difficulty-based sampling	★★★	★
DeepEye-SQL (Li et al., 2025a)	×	✓	✓	all-MiniLM-L6-v2	★★★	★★★
AID-SQL (Li et al., 2025d)	✓	×	N/A	Ranking Model	★★	★★
Derivation-SQL	✓	✓	✓	**DGE**	★★★	★★★

Foundational LLM approaches to Text-to-SQL. Researchers have proposed many novel LLM-based Text-to-SQL approaches. These works have matured into In-Context Learning (ICL) and Fine-Tuning (FT) types. ICL-based methods (Dong et al., 2023; Pourreza & Rafiei, 2023; Talaei et al., 2024) aim to optimize the prompt to handle the task. Chain-of-Thought (CoT) (Wei et al., 2022) and Divide-and-Conquer represent two prevalent strategies for prompt design. CoT methods (Xie et al., 2025; Talaei et al., 2024) encourage LLMs to produce intermediate reasoning steps prior to SQL query generation, whereas Divide-and-Conquer methods (Pourreza & Rafiei, 2023; Pourreza et al., 2025a; Li et al., 2025a) decompose questions into multiple simpler sub-questions, with their solutions collectively forming the final answer. CHESS (Talaei et al., 2024) and Opensearch-SQL (Xie et al., 2025) use CoT to transform natural language questions into SQL queries. DeepEye (Li et al., 2025a) decomposes the task of natural language text-to-SQL into multiple sub-tasks. FT-based approaches (Gu et al., 2023; Li et al., 2024a; 2023b; Yu et al., 2019; Gu et al., 2022; Hoang et al., 2026) are used to fine-tune LLM to gain better performance. XiYan-SQL (Liu et al., 2025b) designs a unique schema representation M-Schema, and fine-tunes a base LLM and an SQL selection model. CHASE-SQL (Pourreza et al., 2025a) fine-tunes an LLM to select the final SQL query from the candidates. *Different from the above research work, our work focuses on improving the performance by exploring the relationship among SQL queries. The similarity between our work and the above research work relies on LLMs. Therefore, their techniques could be directly used in our method. For example, we use Divide-and-Conquer strategy to design the prompt for SQL query generation.*

Few-shot-based Text-to-SQL. In this paper, we review few-shot-based Text-to-SQL focusing on example retrieval. The most prominent feature of these approaches is to retrieve correlated examples to guide LLMs in SQL generation. Table 8 illustrates the differences among these approaches. Recent few-shot-based methods mainly focus on three key dimensions of examples: mining the training set, incorporating external data, and leveraging LLMs to synthesize SQL queries. We summarize these approaches according to these dimensions. ASTRES (Shen et al., 2024), PET-SQL (Li et al., 2024b), MCS-SQL (Lee et al., 2025), DAIL-SQL (Gao et al., 2024), Opensearch-SQL (Xie et al., 2025), QDecomp (Tai et al., 2023) and DeepEye (Li et al., 2025a) retrieve examples solely from the original training set, which limits the diversity of examples. They neither incorporate external information nor leverage LLM-generated SQL queries as examples, thereby ensuring the reliability of the information. Additionally, their processing is entirely offline, utilizing Approximate Nearest Neighbor Search (ANNS) for example retrieval, which keeps computational costs low. In contrast, AID-SQL (Li et al., 2025d) utilizes a retrieval strategy based on a dedicated ranking model, to address the issue where few-shot examples fail to genuinely aid SQL generation. It builds an external knowledge base and trains a ranking model to retrieve examples online. The ranking model makes AID-SQL consume more computational cost than the aforementioned methods. SQLFLOW (Cai

et al., 2025) designs a novel data augmentation method based on the training set. It uses LLM to generate examples based on external databases. *In contrast to the above research work, our work pays attention to the NL question embedding. The proposed Derivation-SQL enhances the diversity of examples by mining new SQL queries from the training set. It does not introduce external information or use LLMs to synthesize SQL queries. Moreover, the entire process is conducted offline and supports the use of ANNS for example retrieval, thereby ensuring high efficiency.*

SQL Decomposition. Many research works propose methods to decompose SQL queries. STEPS (Tian et al., 2023) designs a rule-based method to decompose an SQL query into individual clauses. PARSQL (Dai et al., 2025) designs a parser to decompose an SQL query into sub-SQL queries and reasoning paths. LEARNAT (Liao et al., 2026) decomposes SQL queries into sub-SQL queries with sub-tasks by using LLMs and Monte Carlo Tree Search. While above works have weaknesses. STEPS could not synthesize sub-SQL queries and could not handle complex SQL queries. PARSQL only decomposes atomic conditions and does not break down structural parts like nested SQL queries into smaller pieces. LEARNAT utilizes LLMs to decompose SQL queries, but this approach results in higher computational consumption, and the sub-SQL queries might not follow the given SQL query. Compared with them, Derivation-SQL designs a heuristic approach that achieves high efficiency and success rate in analyzing SQL derivation relationships, and has the capability to handle highly complex SQL queries.