AutoLink: Autonomous Schema Exploration and Expansion for Scalable Schema Linking in Text-to-SQL at Scale

Ziyang Wang[1*], Yuanlei Zheng[1*], Zhenbiao Cao[1], Xiaojin Zhang[2], Zhongyu Wei[3], Pei Fu[4], Zhenbo Luo[4], Wei Chen[1†], Xiang Bai[1]

[1]School of Software Engineering, Huazhong University of Science and Technology
[2]School of Computer Science and Technology, Huazhong University of Science and Technology
[3]School of Data Science, Fudan University
[4]MiLM Plus, Xiaomi Inc.
wangziyang@hust.edu.cn, lemuria_chen@hust.edu.cn

Abstract

For industrial-scale text-to-SQL, supplying the entire database schema to Large Language Models (LLMs) is impractical due to context window limits and irrelevant noise. Schema linking, which filters the schema to a relevant subset, is therefore critical. However, existing methods incur prohibitive costs, struggle to trade off recall and noise, and scale poorly to large databases. We present **AutoLink**, an autonomous agent framework that reformulates schema linking as an iterative, agent-driven process. Guided by an LLM, AutoLink dynamically explores and expands the linked schema subset, progressively identifying necessary schema components without inputting the full database schema. Our experiments demonstrate AutoLink's superior performance, achieving state-of-the-art strict schema linking recall of **97.4%** on Bird-Dev and **91.2%** on Spider 2.0-Lite, with competitive execution accuracy, i.e., **68.7%** EX on Bird-Dev (better than CHESS) and **34.9%** EX on Spider 2.0-Lite (ranking 2nd on the official leaderboard). Crucially, AutoLink exhibits **exceptional scalability**, **maintaining high recall**, **efficient token consumption**, and **robust execution accuracy** on large schemas (e.g., over 3,000 columns) where existing methods severely degrade—making it a highly scalable, high-recall schema-linking solution for industrial text-to-SQL systems.

Code — https://github.com/wzy416/AutoLink

Introduction

Text-to-SQL translates natural-language questions into executable SQL over a given database schema, lowering the barrier for non-experts (Katsogiannis-Meimarakis and Koutrika 2023; Li et al. 2024a). Recent systems rely on autoregressive LLMs: the question and a structured schema representation (e.g., *table/column names*, *descriptions*, and *primary/foreign keys*) are fed into the model, which then generates the SQL sequence (Shi et al. 2025; Hong et al. 2025). However, in large industrial databases, supplying the entire schema S_{full} introduces **substantial noise from irrelevant elements** and **the risk of exceeding context window limits**, hindering correct SQL generation (Lei et al. 2025).

To address these limitations, ***Schema Linking*** emerges as a critical sub-task. Schema linking aims to identify a relevant subset of schema elements ($S_{\text{linked}} \subset S_{\text{full}}$) that are necessary to answer the user's question, thereby reducing the input context and mitigating noise for the subsequent SQL generation module (Wang et al. 2020). The effectiveness of schema linking is often measured by its **strict recall rate (SRR)** (Cao et al. 2024), defined as the proportion of ground-truth schema elements that are successfully included in S_{linked}. A high SRR is paramount, as missing essential schema elements directly limits the upper bound of SQL generation accuracy.

Existing schema linking methods include discriminative scoring of individual tables or columns (e.g., cross-encoders (Li et al. 2023a) and LLM-based scoring (Talaei et al. 2024)), whole-schema reasoning and selection (e.g., full-schema prompting and backward or two-stage pipelines (Lee et al. 2025; Yang et al. 2024)), graph-based modeling of question–schema structure (Li et al. 2023b), and dual-encoder retrieval for fast candidate generation. However, these approaches suffer from scalability limitations in industrial settings, where **computation** and **context windows** become bottlenecks. Achieving high SRR often requires large candidate sets, which reintroduce noise and inflate token usage.

To overcome these challenges, we introduce **AutoLink**, a novel schema linking method that redefines the problem as an interactive, sequential discovery process. Inspired by human database engineers' exploratory workflow, AutoLink employs a large language model powered autonomous agent to dynamically identify and progressively build the relevant schema subset. Crucially, the agent operates without requiring input of the entire database schema. It achieves this by interacting with two specialized environments: one for *direct database exploration* and another for *efficient semantic schema search*. Through a multi-turn dialogue, the agent strategically utilizes a diverse set of actions, including *schema exploration*, *semantic retrieval*, *schema verification*, and *schema expansion*—to iteratively refine the linked

[*]These authors contributed equally.
[†]Corresponding author.

schema. This iterative and exploratory approach enables AutoLink to accurately pinpoint necessary schema elements while effectively filtering out irrelevant information, providing a highly recall schema for subsequent SQL generation.

We rigorously evaluate **AutoLink** on large-scale text-to-SQL benchmarks, Spider 2.0-Lite and Bird dataset. Results show that AutoLink substantially advances schema linking, achieving state-of-the-art SRR with superior token efficiency. In particular, AutoLink attains an SRR of **91.2%** on Spider 2.0-Lite, significantly outperforming all baselines while maintaining high recall. This strong scalability is accompanied by the lowest average token usage across database scales, enabled by iterative schema expansion. Moreover, we observe a strong correlation between SRR and downstream SQL execution accuracy (EX). AutoLink consistently achieves competitive EX performance (e.g., **34.92%** on Spider 2.0-Lite and **68.71%** on Bird), often with substantially fewer tokens, such as using less than half the tokens of leading methods on Spider 2.0-Lite. An extended version with full technical appendices will be released on arXiv.

Related Work

Prior work on schema linking falls into two broad families. ***Element-level schema linking*** scores individual tables or columns for a given question, typically via cross-encoders or LLM-based rankers; representative methods include RESDSQL (Li et al. 2023a), CodeS (Li et al. 2024b), and CHESS (Talaei et al. 2024). ***Database-level schema linking*** reasons over the entire schema and the question, with three common lines: **(i) full-schema prompting and multi-prompt aggregation**—DIN-SQL (Pourreza and Rafiei 2023), MCS-SQL (Lee et al. 2025), C3 (Dong et al. 2023), DAIL-SQL (Gao et al. 2024), E-SQL (Caferoğlu and Özgür Ulusoy 2024), Distillery-SQL (Maamari et al. 2024), Solid-SQL (Liu et al. 2025), TA-SQL (Qu et al. 2024), Reasoning-SQL (Pourreza et al. 2025b), SQL-R1 (Ma et al. 2025); **(ii) backward schema linking**—SQL-to-Schema (Yang et al. 2024), RSL-SQL (Cao et al. 2024); and **(iii) graph-based** approaches—RAT-SQL (Wang et al. 2020), LGESQL (Cao et al. 2021), SADGA (Cai et al. 2021), S^{2}SQL (Hui et al. 2022), ISESL-SQL (Liu et al. 2022), ShadowGNN (Chen et al. 2021), SchemaGraph-SQL (Safdarian et al. 2025).

Building on the above taxonomy, deploying schema linking at industrial scale exposes three recurring bottlenecks. First, **context limitations**: database-level methods often exceed LLM context windows and inflate computation on large schemas. Second, **inefficiency**: element-level methods require $O(|S|)$ scoring passes, which become impractical as $|S|$ grows. Third, a **recall–noise trade-off**: dual-encoder retrievers achieve high strict recall mainly b8y returning many candidates, reintroducing irrelevant schema elements and negating token savings.

Method

To address the limitations of traditional schema linking methods in large industrial databases, we draw inspiration from the **exploratory** and **interactive** workflows of human database engineers. Instead of memorizing an unfamiliar database, they typically locate information through multi-step targeted SQL queries and semantic search. Inspired by this pragmatic approach, in this paper, we reframe schema linking as an interactive, sequential decision-making problem. We propose **AutoLink**, a novel schema linking method built around an autonomous agent powered by a Large Language Model (LLM), tasked with dynamically exploring and expanding the linked schema subset ($S_{\textbf{linked}}$) for given natural language question (Q) **without ever seeing the full database schema** ($S_{\textbf{full}}$). This process is modeled as the agent, guided by an LLM policy (π), executing a sequence of actions within external environments ($\mathcal{E}$), with the aim of maximizing the strict recall of ground-truth schema elements based on the interaction trajectory. The overall framework of our proposed method is illustrated in Figure 1.

External Environment $\mathcal{E}$

To enable agent probing, a pre-built environment ($\mathcal{E}$) is provided for each unique database, comprising two distinct, complementary components: the *Database Environment* $\mathcal{E}_{\text{DB}}$ and the *Schema Vector Store Environment* $\mathcal{E}_{\text{VS}}$.

Env1: Database Environment ($\mathcal{E}_{\textbf{DB}}$) This environment is the live database itself, providing a direct interface for schema and data exploration via SQL execution. It is defined as a function that maps an input SQL query to a formatted execution result:

$$\mathcal{E}_{\text{DB}} : \text{SQL} \rightarrow R_{\text{SQL}}, \tag{1}$$

the output R_{SQL} is a structured textual response designed for exploratory operations. The R_{SQL} output dynamically provides either execution results (truncated if the number of rows exceeds 5) for successful query, or specific error / timeout messages for failures, enabling clear and actionable agent feedback.

Env2: Schema Vector Store Environment ($\mathcal{E}_{\textbf{VS}}$) To facilitate efficient semantic search, we construct a vector database of all columns in the schema. For each column $c_i \in S_{\text{full}}$, we create a textual document by concatenating its core metadata: the column name, parent table's name, data type, and description (if available). We then use a powerful text encoder, i.e., bge-large-en-v1.5 (Chen et al. 2024), to compute a dense vector representation for each column document, which is indexed into a vector store $\mathcal{V}$.

This environment is designed to bridge the semantic gap between a natural language concept and concrete schema elements. Its function is defined as:

$$\mathcal{E}_{\text{VS}} : (q_{\text{nl}}, K, \mathcal{C}_{\text{excl}}) \xrightarrow{\text{retrieve top-}K\text{ columns}} S_{\text{subset}}, \tag{2}$$

the environment takes a natural language query q_{nl}, a retrieval count K, and a set of already retrieved columns $\mathcal{C}_{\text{excl}}$ to avoid redundancy. It performs an Approximate Nearest Neighbor (ANN) search (Liu et al. 2004) within the vector space of columns not in $\mathcal{C}_{\text{excl}}$. While the search identifies the top-K most relevant columns, the output S_{subset} is a **fully structured schema snippet**. This snippet is constructed by gathering all metadata for the retrieved columns, grouping

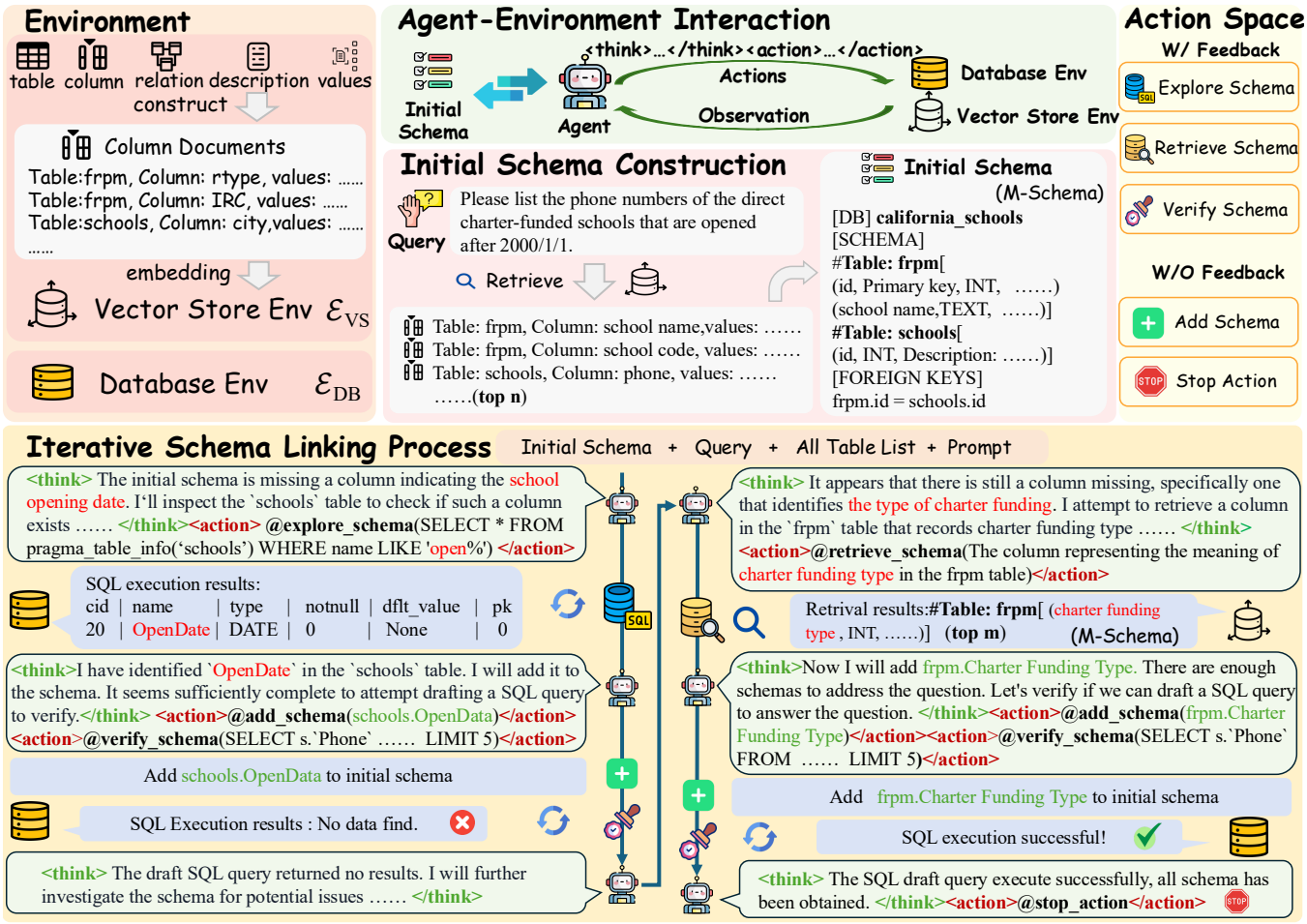

Figure 1: Overview of AutoLink: iterative interactive agentic schema linking framework.

them by their parent tables, and formatting them in a human-readable **M-Schema** style (Gao et al. 2025) that includes data types, keys, and sample values.

Agent-Environment Interaction

Our LLM agent, acting as the decision-making policy π, engages the environment $\mathcal{E}$ in a multi-turn dialogue, driven by the interaction history H. This policy is entirely **prompt-based (training-free)**. The objective is to progressively add potential relevant schema elements for a given user question, obtaining the final linked schema $S_{\text{linked}} \subset S_{\text{full}}$. The process commences with an initial context H_0. To effectively initiate the multi-turn exploration, the agent's first-turn input H_0 is meticulously constructed by concatenating four critical components:

1) Instruction Prompt (I) High-level instructions defining the agent's goal and available actions.

2) User Question (Q) The original user query.

3) Complete Table List (T) All table names in the database (excluding any columns, the total number of tables is typically manageable, e.g., < 100), which provides essential structural context to the agent (the agent needs to know which tables it can query).

4) Initial Schema ($S^{(0)}_{\text{linked}}$) This component is generated by a **single, non-agent-driven call** to the schema vector store environment $\mathcal{E}_{\text{VS}}$. We use the original user question Q as query to retrieve an initial set of candidate columns, creating the initial schema $S^{(0)}_{\text{linked}} = \mathcal{E}_{\text{VS}}(Q, n, \emptyset)$, where n **is a relatively large hyperparameter** (e.g., 50 or 100) to ensure a broad, albeit potentially incomplete, initial set of schema elements highly relevant to the user's question. **The trade-off between recall and noise will be examined in the experimental section**. This provides the agent with crucial structural context beyond just table names, facilitating subsequent decision-making. S_{linked} is also initialized directly with $S^{(0)}_{\text{linked}}$, i.e., $S_{\text{linked}} = S^{(0)}_{\text{linked}}$, and will then be updated throughout the interaction.

In **each subsequent turn** t, the agent generates a reasoning trace θ_t (enclosed within `<think></think>` tags) and a set of actions A_t (within `<action></action>` tags) based on the current history H_t:

$$(\theta_t, A_t) = \pi(H_t). \tag{3}$$

The environment $\mathcal{E}$ then executes these actions, yielding

an observation O_t:

$$O_t = \mathcal{E}(A_t). \tag{4}$$

This resulting triplet (θ_t, A_t, O_t), representing a full turn of interaction, is appended to the history to form H_{t+1}. This loop continues until the agent terminates the process.

Action Space and Agentic Schema Linking

The agent's action space $\mathcal{A}$ provides a set of specialized tools for exploration, verification, and state management. These actions are divided into two categories based on their interaction with the environment.

Actions with Feedback These are the agent's primary tools for gathering new information from the environment.

1. @explore_schema This action aims to interact with the Database Environment ($\mathcal{E}_{\text{DB}}$) by executing exploratory SQL queries, primarily targeting schema metadata or small samples of data, **rather than directly answering the user's question**. For example, the agent can query all columns that contain certain characters (e.g., *id* or *name*) with fuzzy matching, column descriptions and sample column values for certain columns, or examine a table's primary or foreign keys, and other structural metadata, etc.

2. @retrieve_schema This action directly explores missing schema elements within the Schema Vector Store Environment ($\mathcal{E}_{\text{VS}}$). Unlike simple user-question rewrite like CHESS (Talaei et al. 2024), it provides a strong signal for missing schema search. Leveraging the user's natural language question, known table names, and the currently incomplete schema, the agent can directly **infer required missing *column names*, *descriptions* or *concepts* as new query action**. This approach is especially powerful for high-level or ambiguous user queries, narrowing the semantic search space and enabling the discovery of specific, relevant columns beyond simple rephrasing.

Specifically, this action generates a set of retrieved candidate schema elements as: $S_{\text{retrieved}} = \mathcal{E}_{\text{VS}}(q, m, \mathcal{C}_{\text{excl}})$, where m **is a relatively small number** (e.g., 3 or 5) since this action is a targeted retrieval. Previously retrieved columns (from the initial schema or prior **@retrieve_schema** calls) are excluded from the search space.

3. @verify_schema This action interacts with $\mathcal{E}_{\text{DB}}$ and functions as a holistic hypothesis test by attempting to **execute a full SQL query specifically designed to answer the user's question**. The primary purpose is **not to generate the final query result**, but to check for schema sufficiency. A successful execution may indicate completeness, while an error provides a strong, direct signal about which specific schema elements are still missing.

Actions without Feedback These actions manage the agent's internal state and control the workflow.

4. @add_schema This action serves as the agent's mechanism for explicitly committing newly discovered relevant schema elements and updating the linked schema S_{linked}, which we call **Schema Expansion**. After actions with feedback yields new, relevant schema elements, the agent can adopt this action. To improve schema linking recall, we particularly encourage the agent to add schema returned by the **@retrieve_schema** action.

The agent's output for this action specifies the schema elements to be added, presented as a semicolon-separated list of *table_name.column_name* strings within the *add_schema()*. Upon execution, the system processes these identifiers by collecting their complete meta-information. Let S_{added} denote the set of these fully described schema elements. S_{linked} is then updated by merging these newly added schema elements:

$$S_{\text{linked}} \leftarrow S_{\text{linked}} \cup S_{\text{added}}. \tag{5}$$

It is important to note that while the agent can output one or more actions per turn, @add_schema cannot be the sole action, since outputting it alone would leave the agent without environmental feedback for subsequent turns. Therefore, the instruction prompt explicitly requires **@add_schema** to always be paired with at least one feedback-providing action or **@stop_action**.

5. @stop_action This action terminates the multi-turn interaction process. The agent uses this action when, based on its analysis of the dialogue history (including initial schema and expanded schema elements through previous **@add_schema** actions), it determines that enough information has been gathered to answer the user's question. This decision primarily relies on the agent's contextual understanding and significantly influenced by the outcomes of the **@verify_schema** action. Additionally, the process is also terminated if the number of interaction turns exceeds a predefined maximum (**10 turns** in this paper), serving as a safeguard against endless loops.

SQL Generation

With the final linked schema S_{linked} obtained through our iterative, agent-driven process, the subsequent step is **SQL generation**. It is important to note that SQL generation itself is **not the core focus** of this paper, and we leverage existing techniques for this phase. We employ an existing LLM as the SQL generation policy (π_{SQL}). Specifically, given the user's original natural language question Q and final linked schema S_{linked}, the policy adopts a self-consistency strategy (Wang et al. 2023) to sample multiple SQL candidates, then performs syntactic correction (Talaei et al. 2024; Pourreza et al. 2025a) and majority voting (Deng et al. 2025) to derive the final SQL statement.

Experiment

Implementation Details

Experimental Setup Considering the cost-effectiveness of the DeepSeek series model, DeepSeek-V3 serves as the LLM policy π for schema linking (AutoLink), and DeepSeek-R1 and DeepSeek-V3 are employed as the policy π_{SQL} for the subsequent SQL generation phase. The reasoning LLM (DeepSeek-R1) is not chosen for AutoLink's policy due to observed **instruction following degradation** (Fu et al. 2025), which occasionally leads to deviations from the required thought and action format. The **key hyperparameters** for our AutoLink framework include top-n for initial schema retrieval (varied between 5 and 100 during experiments); top-m for the **@retrieve_schema** action, fixed at 3; and a maximum interaction turn limit, set to 10.

Method	Full Input	Bird Dev			Spider2.0-Lite					
		SRR$^{\uparrow}$	$\bar{C}^{\downarrow}$	Avg. Tokens$^{\downarrow}$	SRR$^{\uparrow}_{all}$	SRR$^{\uparrow}_{bq}$	SRR$^{\uparrow}_{sf}$	SRR$^{\uparrow}_{local}$	$\bar{C}^{\downarrow}$	Avg. Tokens$^{\downarrow}$
DE-SL (BGE-Large)	✓	35.5	35.7	–	43.6	28.2	57.1	87.5	153.8	–
CE-SL (BGE-reranker)	✓	72.4	35.7	–	57.6	42.3	72.6	**95.8**	153.8	–
MCS-SQL	✓	85.7	7.5	29.8K	58.9	57.8	54.8	79.2	45.1	168.9K
SQL-to-Schema	✓	92.5	13.5	19.4K	64.0	64.8	54.8	91.7	49.0	171.9K
CHESS	✓	89.7	**4.5**	–	–	–	–	–	–	–
RSL-SQL	✓	93.3	13.0	14.8K	52.0	52.8	44.1	75.0	25.8	29.2K
LinkAlign	✗	–	–	–	36.4	22.5	51.2	66.7	**21.1**	66.7K
AutoLink (ours)	✗	**97.4**	35.8	**8.0K**	**91.2**	**93.7**	**85.7**	**95.8**	159.4	**21.2K**

Table 1: **Performance comparison of schema linking on Bird Dev and Spider 2.0-Lite**. We report the overall Strict Recall Rate (SRR$_{all}$) for both datasets. For Spider 2.0-Lite, SRR is further broken down by SQL dialect: BigQuery (SRR$_{bq}$), Snowflake (SRR$_{sf}$), and SQLite (SRR$_{local}$). $\bar{C}$ denotes the average number of columns included in the simplified schema (S_{linked}) after schema linking. **Full Input** indicates whether the entire database schema is provided to the LLM or if all database schemas are iterated through.

Datasets Our evaluation is conducted on two distinct Text-to-SQL benchmarks: the Spider 2.0-Lite dataset (Lei et al. 2025) and the Bird-Dev dataset (Li et al. 2023c). Bird-Dev comprises 11 databases, featuring an average of 80 columns per database, and includes 1,543 complex SQL query use cases. Spider 2.0-Lite, derived from industrial applications, is designed to reflect the scale of real-world databases. It presents a greater challenge than Bird-Dev, with its databases averaging over 800 columns, more intricate SQL queries, and support for multiple SQL dialects (including BigQuery, Snowflake, and SQLite). This dataset contains 547 test cases.

Evaluation Metrics Model performance is evaluated using 3 primary metrics. The **Strict Recall Rate (SRR)** (Cao et al. 2024) measures the effectiveness of schema linking, defined as the proportion of test cases where the final simplified schema fully contains all required gold schemas. For SQL generation, we report **Execution Accuracy (EX)**, consistent with the official definitions of the Spider and BIRD benchmarks, which measures the consistency between the execution results of predicted and gold SQL queries. Lastly, for LLM-based methods, we report **Avg. Token Consumption**, representing the average total tokens (input plus output) per example. This metric serves as a key metric of computational cost, as overall inference latency is highly susceptible to external factors (e.g., API, network variability), and the time spent on SQL execution within our method is negligible compared to LLM decoding.

Baselines To evaluate AutoLink's performance, we compare it against a diverse set of established methods across both schema linking and SQL generation tasks. For element-level schema linking, our baselines include **DE-SL**, which employs a dual-encoder (BGE-Large-v1.5) to pre-cache column documents and retrieve the top-K most similar columns, and **CE-SL**, which uses a cross-encoder (BGE-reranker) to compute and rank the similarity between the user query and each column individually for top-K retrieval. Additionally, **CHESS** (Talaei et al. 2024) is included, an LLM-based method that scores the user query against each column, followed by sequential table and column filtering. For database-level schema linking, we consider **MCS-SQL** (Lee et al. 2025), which leverages an LLM to directly output relevant schema elements from the full schema and user query, utilizing 5-time sampling decoding to merge results. Similarly, **SQL-to-schema** (Yang et al. 2024) uses an LLM to generate an SQL statement from which involved tables and columns are parsed, also employing 5-time sampling decoding. **RSL-SQL** (Cao et al. 2024) is a hybrid approach combining elements of MCS-SQL and SQL-to-schema, typically with a single decoding pass. Furthermore, **LinkAlign** (Wang, Liu, and Yang 2025) stands as a baseline for its query rewriting and multi-agent discussion framework that also bypasses the need for schema element retrival.

Main Results of Schema Linking

As shown in Table 1, all models (except for **DE-SL** and **CE-SL**) are implemented using DeepSeek-V3 as the backbone to ensure fair and consistent comparison. For the hyperparameter top-n for initial schema retrieval, we set 30 on Bird Dev and 100 on Spider 2.0-Lite. The experimental results demonstrate that **AutoLink** achieves a significant advantage over baseline methods in terms of SRR and average token consumption across both the Bird and Spider 2.0-Lite. On the more challenging Spider 2.0-Lite benchmark, our method achieves a **27.2%** improvement in strict recall rate compared to the second place (**SQL-to-Schema**), while reducing the maximum token consumption by **87.7%**. Compared with encoder-based methods (**DE-SL** and **CE-SL**), although the number of recalled columns is close, our method improves SRR by an average of approximately **40%**.

Compared to methods requiring full schema input, such as **MCS-SQL**, **SQL-to-Schema**, and **RSL-SQL**, we observe that while they perform acceptably on smaller datasets like Bird, their performance sharply declines on larger, more complex databases like Spider 2.0-Lite. This is due to the long context lengths and high resource consumption at scale. Although these methods can theoretically enhance SRR by increasing sampling decoding iterations, we found that the

Method	Model	EX↑	Avg. Tokens↓
Spider-Agent	QwQ	11.33	–
	GPT-4o	13.16	–
	DeepSeek-R1	13.71	–
	o1-preview	23.03	–
	o3-mini	23.40	–
	Claude-3.7-Sonnet	28.52	–
CHESS	GPT-4o	3.84	–
LinkAlign	DeepSeek-R1	33.09	–
RSL-SQL†	DeepSeek-R1	30.53	50.0K
REFORCE†	DeepSeek-R1	29.62	81.1K
REFORCE	o1-preview	30.35	81.1K
REFORCE	GPT-o3	**37.84**	81.1K
AutoLink	DeepSeek-R1	34.92	**38.0K**

Table 2: Execution Accuracy (EX) comparison of different methods on the Spider 2.0-Lite dataset. † indicates results obtained through independent reproduction. Avg. Tokens indicates the average number of tokens comsumed to generate a SQL.

Method	Model	EX↑	Avg. Tokens↓
MAC-SQL	GPT-4	59.39	**6.8K**
TA-SQL	GPT-4	56.19	7.3K
RSL-SQL	GPT-4o	67.21	7.5K
CHESS	Gemini-1.5-Pro	68.31	14.5K
AutoLink	DeepSeek-v3	66.36	8.0K
AutoLink	Gemini-1.5-Pro	**68.71**	8.0K

Table 3: Execution Accuracy (EX) comparison of different methods the Bird Dev.

Method	$SRR_{n=5}$	$SRR_{n=50}$	$SRR_{n=100}$
AutoLink	**79.2**	**88.0**	**91.2**
- w/o Verify Schema	$77.2_{\downarrow 2.0}$	$86.8_{\downarrow 1.2}$	$89.6_{\downarrow 1.6}$
- w/o Explore Schema	$76.8_{\downarrow 2.4}$	$84.8_{\downarrow 3.2}$	$88.8_{\downarrow 2.4}$
- w/o Retrieve Schema	$72.4_{\downarrow 6.8}$	$80.4_{\downarrow 7.6}$	$84.5_{\downarrow 6.7}$

Table 4: Ablation study on the impact of different schema linking actions under varying numbers of initial candidates n on Spider 2.0-Lite.

SRR growth becomes negligible and quickly saturates, particularly on the Spider 2.0-Lite. *Blindly increasing decoding attempts yields minimal performance gains*, as the results across different decoding passes often exhibit low diversity. Consequently, **these methods struggle to effectively control the number of recalled columns to achieve a comparable scale to ours**. In contrast, our method consistently maintains strong performance across varying dataset scales. We demonstrate the scatter plot of recalled columns versus SRR in extended version, and our approach still **achieves superior recall rates even when recalling a similar small number of columns**. Regarding the comparison of Token consumption, we found that compared with above methods, our method significantly reduces the Token overhead. Compared with RSL-SQL, although its Token overhead is not large, its recall performance in large-scale databases is relatively low.

In contrast to methods like **CHESS** and **LinkAlign**, which **prioritize noise reduction by aiming to include only the columns strictly required by the gold SQL**, our approach differs. This noise-minimization strategy carries substantial risk, as evidenced by LinkAlign's mere 36.4% SRR, despite its minimal average of 21.1 recalled columns. Such a low recall rate severely compromises the utility of the simplified schema for subsequent SQL generation.

Results of SQL Generation

The comparative results in Table 2 and Table 3 demonstrate that **AutoLink** achieves competitive EX on both Spider 2.0-Lite and Bird. On Spider 2.0-Lite, AutoLink attains an EX score of **34.92%** using DeepSeek-R1, which outperforms most baseline approaches and is highly competitive with the best-performing method, ReFoRCE (37.84% with GPT-o3). Notably, ReFoRCE generates eight candidate SQL queries per example, whereas AutoLink generates only five. This difference results in a significant advantage for AutoLink in terms of token consumption—AutoLink requires only **38.0K** tokens on average, less than half of ReFoRCE's 81.1K. When evaluated using the same model, DeepSeek-R1, our method achieves the best performance. Similarly, on the Bird dev set, AutoLink also achieves strong results. With Gemini-1.5-Pro as the backbone model, AutoLink achieves an EX score of **68.71%**, which is competitive with or superior to strong baselines such as **CHESS** and **RSL-SQL**.

Ablation Study

The ablation results in Table 4 demonstrate the importance of each core agent action for schema linking under initial retrieval sizes of top-5, top-50, and top-100. Removing any single action—**@retrieve_schema**, **@explore_schema**, or **@verify_schema**—leads to a clear drop in SRR across all initial top-n settings. Notably, removing schema retrieval has the largest impact, highlighting its key role in identifying relevant columns. Without this capability, the agent is much less able to anchor its reasoning in the most promising schema elements. Excluding database exploration also consistently reduces performance, underscoring its value for resolving ambiguities when facing unfamiliar databases. Although omitting verification results in a smaller decrease in SRR, its contribution is evident across all candidate sizes, reflecting the importance of continual self-assessment within the agent's reasoning process.

Scalability Analysis of Different Database Scales

Figure 2 compares the SRR of various schema linking methods: **AutoLink**, **BGE-Large (DE-SL)**, **BGE-reranker (CE-SL)**, and **LinkAlign**, etc., on Spider 2.0-Lite across increasing database scales. All methods exhibit a significant decline in SRR as the database size grows, though the decrease of **AutoLink** is notably slower. In large databases

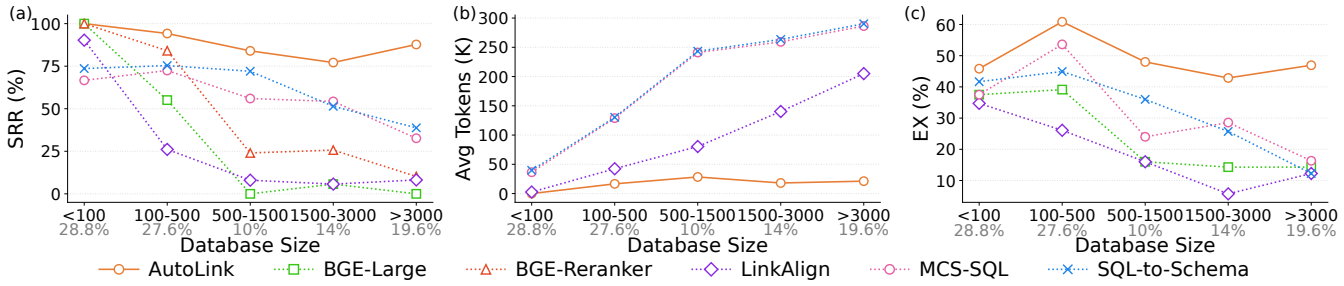

Figure 2: Scalability comparison across databases on Spider 2.0-Lite of varying sizes in terms of (a) Strict Recall Rate (SRR↑), (b) Average Tokens Consumption (Avg. Tokens↓), and (c) Execution Accuracy (EX↑). Percentages below each bin indicate the proportion of databases within each size range.

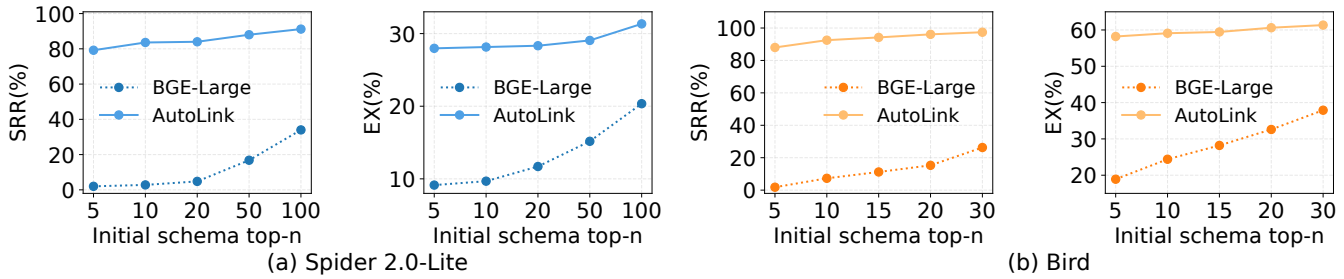

Figure 3: SRR and EX comparison of schema linking at different initial top-n on Spider 2.0-Lite and Bird dev set.

exceeding 3,000 columns, the SRR of all baseline models drops below **40%**, yet AutoLink's SRR remains near **90%**. For token consumption ((b) in Figure 2), all methods generally show an upward trend. AutoLink consistently demonstrates the lowest average token usage across all database sizes. This efficiency stems from AutoLink's iterative approach, which begins with a smaller schema and gradually expands it, resulting to minimal changes in token consumption across different database scales.

To ensure a fair comparison, our SQL generation method is applied to the linked schemas produced by different schema linking approaches. Regarding EX ((c) in Figure 2), the performance of all baselines declines as schema size increases. Crucially, we observe a direct correlation: **models with higher SRR tend to exhibit higher EX**. This highlights that a higher SRR is instrumental in improving the EX of subsequent SQL generation. This is intuitively sound because if the SQL generator (i.e., the LLM), does not hallucinate, it will only utilize the schema elements provided by the schema linking step. Consequently, an incomplete schema (due to low SRR) means the generated SQL may lack necessary tables and columns from the gold SQL, making it highly probable that the generated query will be incorrect. **AutoLink** consistently achieves the highest EX across all size ranges, significantly outperforming baselines on the largest databases due to its superior SRR.

Analysis of Hyperparameter

As shown in Figure 3, **AutoLink** consistently outperforms BGE Large in SRR under various initial top-n schema retrieval settings. Increasing top-n improves SRR for both methods, but AutoLink's agent-driven exploration gives it a notable advantage even **at small top-n values** (e.g., top-5) by effectively identifying and incorporating missing schema components. This robustness holds across datasets of different scales, demonstrating good scalability.

For downstream SQL EX on Spider 2.0-Lite, AutoLink consistently achieves higher accuracy than BGE across all top-n settings. While its performance rises with larger top-n, the improvement plateaus at high values, suggesting that **most critical schema elements can already be recovered with smaller candidate sets**. The advantage is most pronounced at low top-n, highlighting AutoLink's effectiveness when initial schema retrieval is incomplete.

Conclusion

In this paper, we propose AutoLink, a novel framework that redefines schema linking as an adaptive, agent-driven process. By orchestrating semantic retrieval from a vector store and utilizing lightweight SQL probes, our LLM-powered agent iteratively and autonomously assembles only the schema elements truly necessary for a given query, without requiring the full database schema as input. This unified mechanism achieves state-of-the-art strict recall on Spider 2.0-Lite and Bird, significantly reduces token usage, and demonstrates exceptional scalability and robustness on large schemas with thousands of columns, thereby providing a practical and efficient foundation for industrial-scale Text-to-SQL systems.

Acknowledgments

This research was supported by *National Natural Science Foundation of China* (No.62406121) and *National Science Foundation of Hubei Province, China* (No.2024AFB189).

References

Caferoğlu, H. A.; and Özgür Ulusoy. 2024. E-SQL: Direct Schema Linking via Question Enrichment in Text-to-SQL. arXiv:2409.16751.

Cai, R.; Yuan, J.; Xu, B.; and Hao, Z. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34: 7664–7676.

Cao, R.; Chen, L.; Chen, Z.; Zhao, Y.; Zhu, S.; and Yu, K. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In Zong, C.; Xia, F.; Li, W.; and Navigli, R., eds., *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2541–2555. Online: Association for Computational Linguistics.

Cao, Z.; Zheng, Y.; Fan, Z.; Zhang, X.; Chen, W.; and Bai, X. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. arXiv:2411.00073.

Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; and Liu, Z. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics: ACL 2024*, 2318–2335. Bangkok, Thailand: Association for Computational Linguistics.

Chen, Z.; Chen, L.; Zhao, Y.; Cao, R.; Xu, Z.; Zhu, S.; and Yu, K. 2021. ShadowGNN: Graph Projection Neural Network for Text-to-SQL Parser. In Toutanova, K.; Rumshisky, A.; Zettlemoyer, L.; Hakkani-Tur, D.; Beltagy, I.; Bethard, S.; Cotterell, R.; Chakraborty, T.; and Zhou, Y., eds., *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 5567–5577. Online: Association for Computational Linguistics.

Deng, M.; Ramachandran, A.; Xu, C.; Hu, L.; Yao, Z.; Datta, A.; and Zhang, H. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Format Restriction, and Column Exploration. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.

Dong, X.; Zhang, C.; Ge, Y.; Mao, Y.; Gao, Y.; lu Chen; Lin, J.; and Lou, D. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306.

Fu, T.; Gu, J.; Li, Y.; Qu, X.; and Cheng, Y. 2025. Scaling Reasoning, Losing Control: Evaluating Instruction Following in Large Reasoning Models. arXiv:2505.14810.

Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; and Zhou, J. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.*, 17(5): 1132–1145.

Gao, Y.; Liu, Y.; Li, X.; Shi, X.; Zhu, Y.; Wang, Y.; Li, S.; Li, W.; Hong, Y.; Luo, Z.; Gao, J.; Mou, L.; and Li, Y. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599.

Hong, Z.; Yuan, Z.; Zhang, Q.; Chen, H.; Dong, J.; Huang, F.; and Huang, X. 2025. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL . *IEEE Transactions on Knowledge & Data Engineering*, 37(12): 7328–7345.

Hui, B.; Geng, R.; Wang, L.; Qin, B.; Li, Y.; Li, B.; Sun, J.; and Li, Y. 2022. S^2SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers. In Muresan, S.; Nakov, P.; and Villavicencio, A., eds., *Findings of the Association for Computational Linguistics: ACL 2022*, 1254–1262. Dublin, Ireland: Association for Computational Linguistics.

Katsogiannis-Meimarakis, G.; and Koutrika, G. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, 32(4): 905–936.

Lee, D.; Park, C.; Kim, J.; and Park, H. 2025. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. In Rambow, O.; Wanner, L.; Apidianaki, M.; Al-Khalifa, H.; Eugenio, B. D.; and Schockaert, S., eds., *Proceedings of the 31st International Conference on Computational Linguistics*, 337–353. Abu Dhabi, UAE: Association for Computational Linguistics.

Lei, F.; Chen, J.; Ye, Y.; Cao, R.; Shin, D.; SU, H.; SUO, Z.; Gao, H.; Hu, W.; Yin, P.; Zhong, V.; Xiong, C.; Sun, R.; Liu, Q.; Wang, S.; and Yu, T. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations*.

Li, B.; Luo, Y.; Chai, C.; Li, G.; and Tang, N. 2024a. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proc. VLDB Endow.*, 17(11): 3318–3331.

Li, H.; Zhang, J.; Li, C.; and Chen, H. 2023a. RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press. ISBN 978-1-57735-880-0.

Li, H.; Zhang, J.; Liu, H.; Fan, J.; Zhang, X.; Zhu, J.; Wei, R.; Pan, H.; Li, C.; and Chen, H. 2024b. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data*, 2(3).

Li, J.; Hui, B.; Cheng, R.; Qin, B.; Ma, C.; Huo, N.; Huang, F.; Du, W.; Si, L.; and Li, Y. 2023b. Graphix-T5: mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press. ISBN 978-1-57735-880-0.

Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; Zhou, X.; Ma, C.; Li, G.; Chang, K. C.; Huang, F.; Cheng, R.; and Li, Y. 2023c. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Liu, A.; Hu, X.; Lin, L.; and Wen, L. 2022. Semantic Enhanced Text-to-SQL Parsing via Iteratively Learning Schema Linking Graph. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, 1021–1030. New York, NY, USA: Association for Computing Machinery. ISBN 9781450393850.

Liu, G.; Tan, Y.; Zhong, R.; Xie, Y.; Zhao, L.; Wang, Q.; Hu, B.; and Li, Z. 2025. Solid-SQL: Enhanced Schema-linking based In-context Learning for Robust Text-to-SQL. In Rambow, O.; Wanner, L.; Apidianaki, M.; Al-Khalifa, H.; Eugenio, B. D.; and Schockaert, S., eds., *Proceedings of the 31st International Conference on Computational Linguistics*, 9793–9803. Abu Dhabi, UAE: Association for Computational Linguistics.

Liu, T.; Moore, A.; Yang, K.; and Gray, A. 2004. An Investigation of Practical Approximate Nearest Neighbor Algorithms. In Saul, L.; Weiss, Y.; and Bottou, L., eds., *Advances in Neural Information Processing Systems*, volume 17. MIT Press.

Ma, P.; Zhuang, X.; Xu, C.; Jiang, X.; Chen, R.; and Guo, J. 2025. SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning. arXiv:2504.08600.

Maamari, K.; Abubaker, F.; Jaroslawicz, D.; and Mhedhbi, A. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Pourreza, M.; Li, H.; Sun, R.; Chung, Y.; Talaei, S.; Kakkar, G. T.; Gan, Y.; Saberi, A.; Ozcan, F.; and Arik, S. 2025a. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In Yue, Y.; Garg, A.; Peng, N.; Sha, F.; and Yu, R., eds., *International Conference on Representation Learning*, volume 2025, 60385–60415.

Pourreza, M.; and Rafiei, D. 2023. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc.

Pourreza, M.; Talaei, S.; Sun, R.; Wan, X.; Li, H.; Mirhoseini, A.; Saberi, A.; and Arik, S. O. 2025b. Reasoning-SQL: Reinforcement Learning with SQL Tailored Partial Rewards for Reasoning-Enhanced Text-to-SQL. arXiv:2503.23157.

Qu, G.; Li, J.; Li, B.; Qin, B.; Huo, N.; Ma, C.; and Cheng, R. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics ACL 2024*, 5456–5471. Bangkok, Thailand and virtual meeting: Association for Computational Linguistics.

Safdarian, A.; Mohammadi, M.; Jahanbakhsh, E.; Naderi, M. S.; and Faili, H. 2025. SchemaGraphSQL: Efficient Schema Linking with Pathfinding Graph Algorithms for Text-to-SQL on Large-Scale Databases. arXiv:2505.18363.

Shi, L.; Tang, Z.; Zhang, N.; Zhang, X.; and Yang, Z. 2025. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *ACM Comput. Surv.*, 58(2).

Talaei, S.; Pourreza, M.; Chang, Y.-C.; Mirhoseini, A.; and Saberi, A. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7567–7578. Online: Association for Computational Linguistics.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

Wang, Y.; Liu, P.; and Yang, X. 2025. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 977–991. Suzhou, China: Association for Computational Linguistics. ISBN 979-8-89176-332-6.

Yang, S.; Su, Q.; Li, Z.; Li, Z.; Mao, H.; Liu, C.; and Zhao, R. 2024. SQL-to-Schema Enhances Schema Linking in Text-to-SQL. In Strauss, C.; Amagasa, T.; Manco, G.; Kotsis, G.; Tjoa, A. M.; and Khalil, I., eds., *Database and Expert Systems Applications*, 139–145. Cham: Springer Nature Switzerland. ISBN 978-3-031-68309-1.