DoubleAgent: Towards Text-to-SQL on Enterprise Databases

Fares Elkholy*
Technical University of Darmstadt

Timo Eckmann*
Technical University of Darmstadt

Jan-Micha Bodensohn
Technical University of Darmstadt

Mulang' Onando
SAP SE

Carsten Binnig
Technical University of Darmstadt &
hessian.ai & DFKI

ABSTRACT

Recent progress on Text-to-SQL has almost saturated existing benchmarks such as Spider, BIRD, and Spider 2.0-Snow. These benchmarks, however, rest on the assumption that database schemas, albeit complex, are descriptive enough to be understood by LLMs without background knowledge. Real-world enterprise databases break this assumption by naming fields with non human-readable names that are hard to interpret without business knowledge. When using LLMs naively on such enterprise databases, accuracy drops significantly. To close this performance gap, we study how knowledge graphs that explain the semantics of the schema can provide the missing knowledge. Our prototype DoubleAgent, a ReAct agent that can query databases and knowledge graphs *simultaneously* in a single reasoning loop, demonstrates that knowledge graphs can help restore accuracy on enterprise schemas from 10.9% to 43.6%.

VLDB Workshop Reference Format:
Fares Elkholy, Timo Eckmann, Jan-Micha Bodensohn, Mulang' Onando, and Carsten Binnig. DoubleAgent: Towards Text-to-SQL on Enterprise Databases. VLDB 2026 Workshop: Novel Optimizations for Visionary AI Systems (NOVAS).

1 INTRODUCTION

Text-to-SQL has advanced significantly. Text-to-SQL translates natural language requests into executable SQL queries, enabling users to query relational databases without writing SQL by hand. This line of work has seen rapid progress in recent years, driven by a succession of increasingly ambitious benchmarks, from Spider [12] to BIRD [6] to Spider 2.0 [5], each larger and more realistic than the last. Performance has climbed as well, with the strongest approaches surpassing 80% execution accuracy on BIRD and over 90% on the enterprise-scale Spider 2.0-Snow. While these existing benchmarks are almost saturated, the progress on them rests on the quiet assumption that the database schemas, albeit complex in scale, are still understandable without background knowledge since their identifiers read like natural language. In enterprise systems, however, this is often not the case [1].

Enterprise schemas not designed to be understood. Enterprise databases often expose data through non human-readable (i.e., cryptic) identifiers instead of descriptive attribute names. In SAP S/4HANA, for example, an attribute storing the net purchase order value is named `POurgDocNetAmount`, and a delivery date becomes `BSTLDAT`. This is because schemas are often designed merely to hold the data for specific applications rather than to enable future understanding. Existing Text-to-SQL benchmarks do not fully reflect this reality. Spider, BIRD, and Spider 2.0-Snow all use expressive English identifiers that map directly to natural language.

Naively using LLMs does not work. Faced with such cryptic schemas, simply prompting an LLM falls short. As the evaluation of state-of-the-art approaches on the enterprise benchmark BEAVER shows, performance is critically low [2], and we observe the same on a real-world SAP dataset. Importantly, we find that making the schema more enterprise-like by obscuring attribute names while leaving the questions and data untouched is enough to break Text-to-SQL agents like Spider-Agent, with accuracy dropping by roughly 30 percentage points.

Introducing DoubleAgent. While the schemas in enterprise systems are often obscure, there typically exists a trove of business knowledge about these systems that supports their operation and maintenance. Many enterprises have recently begun to build and maintain knowledge graphs that encode this internal knowledge about business entities, domain concepts, and even semantic descriptions of individual attributes. For Text-to-SQL, such knowledge graphs can serve as the missing dictionary that maps cryptic physical identifiers to semantic business concepts in natural language. An agent that can query both sources thus gains access to both the *meaning* of the data (through the knowledge graph) as well as the *data itself* (through the database). Motivated by this insight, we propose the idea of a DoubleAgent that simultaneously issues SQL queries against the relational database and semantic queries against the business knowledge graph. Within a single ReAct [11] agent loop, DoubleAgent navigates the knowledge graph to ground its SQL against the actual meaning of the data.

Contributions. In this paper, we present a prototype of DoubleAgent, a Text-to-SQL agent that combines knowledge graph exploration with SQL execution in a single agent loop. To support further research on the topic, we also present Spider 2.0-Cryptic, a benchmark based on Spider 2.0-Snow that incorporates the cryptic attribute names of real-world enterprise systems. Finally, we evaluate DoubleAgent on Spider 2.0-Cryptic as well as in a real-world SAP S/4HANA setting with 516 OData questions, showing that it can restore accuracy from 10.9% to 43.6% on Spider 2.0-Cryptic and strict accuracy from 31.8% to 46.9% in the real-world setting.

*Equal contribution. Correspondence goes to timo.eckmann@tu-darmstadt.de

Proceedings of the VLDB Endowment. ISSN 2150-8097.

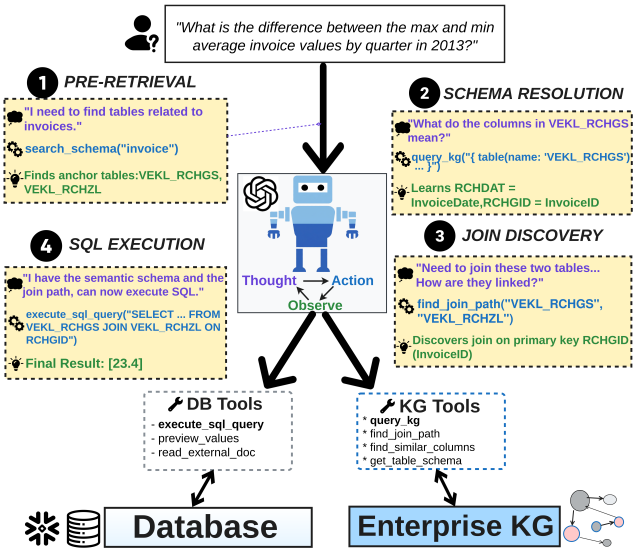

Figure 1: To answer a user question, DoubleAgent first performs a vector-based schema pre-retrieval❶ to identify relevant anchor tables. It then enters a ReAct loop, alternating between querying the enterprise knowledge graph for semantic resolution❷❸ and executing SQL against the database,❹ before finally terminating via the `finish` tool.

2 OUR SOLUTION: A DOUBLE AGENT

On enterprise databases, the hard part is not just translating the query intent into SQL, but also working out what the columns mean. DoubleAgent therefore inverts the usual order. Before it writes a single query, it investigates the existing business knowledge graph to recover the semantic meaning of the obscure schema. This recovered meaning is then turned into SQL against the database.

Finding the right parts of the schema. Before the agent can reason at all, it faces a practical problem: A single database often stores tens of thousands of columns, far more than fit into the LLM's context window. Therefore, the agent must first find the tables and columns that the question actually needs. DoubleAgent thus starts with a schema retrieval step that runs before the agent loop starts (Figure 1). Following HybridRAG [9], it splits the question into short, schema-oriented phrases and embeds them with a sentence encoder [10]. It then uses vector search to find the table and column descriptions in the knowledge graph that are most relevant to the question, which become the anchor points for the agent.

Deciding how much of the schema to show. These anchors give a first idea about which columns matter, but not in what context. A column belongs to a table, and that table may have hundreds of other columns that the agent has to consider as well. How much of the surrounding schema to show to the agent depends on the database: For a small schema that fits into the LLM's context window, DoubleAgent shows it in full and simply marks the anchor columns. A large schema does not fit, and forcing it in would only bury the relevant columns among thousands of irrelevant columns. DoubleAgent therefore reduces the schema to ranked table summaries built around the anchors, and later expands summaries into full column lists on demand with a `get_table_schema` tool.

Reasoning over the knowledge graph and database. With the schema in context, either in full or as summaries the agent can expand, the main problem remains that the names are still cryptic, making it hard to map a question onto them. The agent must first resolve the meaning behind each name and then check its understanding against the data. Neither can be done in a single pass, since the agent often learns what a name means only after probing it. We therefore run a ReAct loop [11] that works both fronts at once. On the knowledge graph side, the agent looks up the business names and descriptions behind cryptic identifiers, loads relevant external documents into context, traces likely joins, and finds related columns. On the database side, it inspects sample values and executes SQL to test whether its mapping actually returns the right data. The agent alternates between both sides until it reaches a result in which it has confidence or exhausts a budget of 25 steps.

Interacting with the knowledge graph and database. DoubleAgent hinges on the tools that enable it to interact with the knowledge graph and database. The knowledge graph tools serve to recover meaning. At any step, the agent can traverse the knowledge graph with GraphQL [4] queries and thereby pull multiple properties such as business names, descriptions, and neighbors in a single call (`query_kg`). Furthermore, because enterprise schemas rarely declare explicit foreign-key relationships [5], the agent can also recover how two tables join by following `LIKELY_FK` edges in the knowledge graph (`find_join_path`), or find columns that store the same thing under a different name by following `SEM_SIMILAR` edges (`find_similar_columns`). The database tools then put this recovered meaning to the test. Since a cryptic schema may also hide how its values are represented, the agent is hinted to first inspect the real data formats (`preview_values`) before querying the database (`execute_sql_query`). One common failure mode are incorrect SQL queries leading to empty results. Therefore, the agent is instructed to treat an empty result as a signal rather than an answer. A repeatedly empty query pushes the agent to inspect values and loosen filters until it finds the data. Finally, the agent must confirm its final result by calling `finish`.

3 OUR KNOWLEDGE GRAPH

As mentioned in the introduction, many enterprises have now begun to build knowledge graphs that encode their internal business knowledge about entities, domain concepts, and even semantic descriptions of individual attributes. For DoubleAgent, such a knowledge graph can provide the semantic context required to ground the SQL, serving as a dictionary that maps cryptic identifiers (e.g., `BSTL_DAT`) back to their business meaning.

Constructing the knowledge graph. To represent an existing enterprise knowledge graph, we construct a synthetic knowledge graph based on the human-readable Spider 2.0-Snow dataset that Spider 2.0-Cryptic is based on. Following existing work on real-world enterprise knowledge graphs [7], we model our knowledge graph as a two-layer property graph: a *cryptic layer*, which preserves the physical schema identifiers (e.g., `LFRNG`, `AUFTR`), and a *semantic layer* that holds their corresponding semantic names, descriptions,

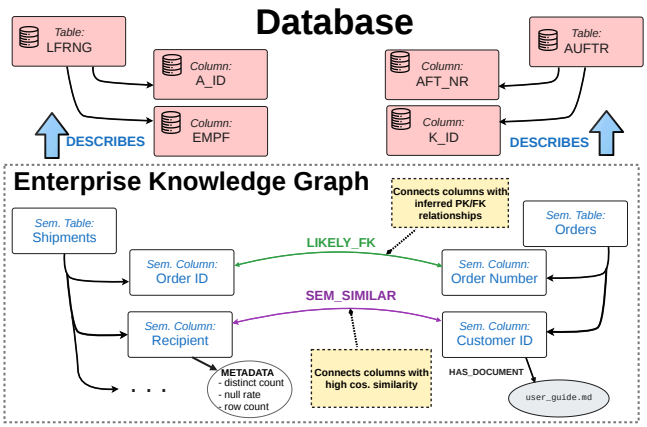

Figure 2: Anatomy of an enterprise knowledge graph. Top: the *cryptic layer* of physical database identifiers. Bottom: the *semantic layer* of semantic schema nodes with descriptions, data profiles, metadata and external documents. Enrichment edges (`LIKELY_FK` & `SEM_SIMILAR`) capture likely joins and semantic similarity.

and data profiles (see Figure 2). `DESCRIBES` edges bind the two layers together, enabling the agent to reason in the semantic space while generating executable SQL against the cryptic schema.

Completing the knowledge graph. Similar to many real-word enterprise schemas, Spider 2.0-Snow does not provide explicit foreign-key/primary-key relationships, and schema metadata is generally sparse. Therefore, we infer this missing context programatically from Spider 2.0-Snow to complete the knowledge graph. To compensate for the sparse metadata, we enrich the graph by generating descriptions for undescribed tables and columns using `GPT-5-Mini`. Moreover, we infer `LIKELY_FK` edges to recover join paths that are not explicitly enforced by the database. Specifically, we extract base entities from foreign-key suffixes (e.g., `_id`, `_fk`) and link them to corresponding primary keys, assigning a confidence score based on naming heuristics, data type equivalence, and value uniqueness.

Furthermore, we introduce `SEM_SIMILAR` edges to link pairs of columns whose underlying meaning (derived from semantic names, data types, and sample values) exhibits high cosine similarity. For instance, as shown in Figure 2, the semantic nodes "Recipient" and "Customer ID" reference the same underlying entity. Making this relationship explicit helps the agent avoid missing key columns during join operations. Finally, we embed every node in the knowledge graph using `all-MiniLM-L6-v2` [10]. This allows the agent to retrieve relevant schema elements via vector search and inject a concise slice into its context, as detailed in Section 2.

4 A NEW TEXT-TO-SQL BENCHMARK

In this section, we describe the enterprise Text-to-SQL setting in more detail and introduce Spider 2.0-Cryptic, the first benchmark that pairs enterprise-scale schemas with cryptic SAP-style attribute names to enable a controlled analysis of the effect such cryptic names have on Text-to-SQL agents.

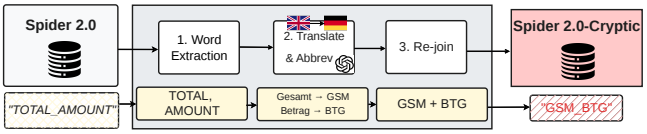

Figure 3: To construct Spider 2.0-Cryptic, each Spider 2.0-Snow identifier is split into words, each word is translated into a German abbreviation, and the abbreviations are re-joined into a cryptic name.

4.1 Why Are Existing Benchmarks Not Enough?

While Text-to-SQL benchmarks are slowly moving towards enterprise reality, no existing benchmark lets us measure the effect of enterprise-like cryptic schema names in isolation, with each benchmark falling short in a different way. Spider 1.0 uses tiny databases with on average only 27.6 columns each [12]. BIRD is larger and noisier, yet its databases still average just 54 columns [6]. Spider 2.0 [5] is the first benchmark to reach enterprise scale, since its 213 public cloud databases average 744 columns and reach into the thousands in the largest cases. However, its identifiers remain descriptive English names. BEAVER instead samples from private enterprise warehouses, so its names are no longer descriptive [2]. Among these benchmarks, it is the most realistic; however, realism and isolation pull in opposite directions as BEAVER varies schema names, scale, data, and domain. Consequently, it becomes impossible to attribute any observed decline in accuracy to a single cause, such as the cryptic schema names. ADVETA provides both a clean as well as a cryptic version of the same schema by perturbing column names [8], but its perturbations are adversarial substitutions on small Spider tables, not the SAP-style abbreviations that span an entire database schema. Therefore, no existing benchmark provides both a clear and a cryptic version of the same enterprise-scale schema to isolate the effect of obfuscation.

4.2 Our New Benchmark: Spider 2.0-Cryptic

The purpose of Spider 2.0-Cryptic is to enable the evaluation of enterprise-style schema obfuscation. To measure this dimension only, we start from Spider 2.0-Snow [5], a benchmark that is already enterprise-scale and readable, and rewrite only its schema names into cryptic SAP-style abbreviations while keeping the databases, questions, and data fixed. This produces a matching pair of schemas, identical in every aspect except their table and column names, making any drop in accuracy attributable to the cryptic schema.

Constructing Spider 2.0-Cryptic. To make the cryptic names in Spider 2.0-Cryptic realistic, we studied a real-world S/4HANA OData dataset at SAP and imitated how its fields are abbreviated. To build Spider 2.0-Cryptic, we first select a 50-database subset of Spider 2.0-Snow [5], focusing on domains that are representative of enterprise data warehouses, such as government filings, healthcare, and e-commerce. Subsequently, as shown in Figure 3, we obfuscate only the schema names by (i) tokenizing every table and column name (e.g., using underscores and case changes as delimiters), which yields 5,288 unique English words, (ii) mapping each word to a German abbreviation of 2 to 6 characters with `GPT-5-Mini`, so that `customer` becomes `KND` and `order` becomes

BSTL, and (iii) re-joining the abbreviations into the final cryptic identifier. We translate to German because SAP's physical names also derive from German business terms. Temporal suffixes such as `_2022` are kept verbatim to preserve semantic partitioning. We then deploy the obfuscated schemas as Snowflake views that alias the original names, so every query runs against the exact same data as Spider 2.0-Snow. Any difference in accuracy is therefore attributable to the schema names, and not to the data.

Spider 2.0-Cryptic in numbers. The resulting Spider 2.0-Cryptic dataset consists of clear and cryptic versions of the same enterprise-scale schemas, differing only in their names. In total, the benchmark spans 50 databases, 2,951 tables, and 259,482 columns, with 276 evaluation questions across 12 domains. The schema sizes are heavily skewed. While the median database holds 218 columns, the mean is around 5,200, and the largest database (`FEC`) reaches 485 tables and 71,819 columns. At this scale, an agent can no longer fit the full schema into its context. Finally, 56 questions (around 20%) are supported by external knowledge documents that contain proprietary domain knowledge and column descriptions.

5 INITIAL EVALUATION

Our initial experiments evaluate DoubleAgent on Spider 2.0-Cryptic and on a real SAP S/4HANA system. The results show that a knowledge graph lets the agent recover most of the accuracy lost due to cryptic schemas.

Evaluation setup. For Text-to-SQL, we use Spider 2.0-Cryptic with its 276 questions over a clear and a cryptic version of the schema. For the transfer study, we use 516 Text-to-OData questions over a real SAP S/4HANA system. On Spider 2.0-Cryptic, we report Execution Accuracy (EX), where a query is correct if its result table matches that of the gold query. On SAP S/4HANA, no live endpoint was available, so we instead report *strict accuracy*, where a generated OData query is correct only if it matches the gold query on all three of its parts, namely the target service and entity set, the filter predicates, and the selected fields. All agents use `GPT-5-Mini` with `high` thinking effort. We evaluate DoubleAgent with and without its knowledge graph, where disabling it removes the knowledge graph tools and the pre-retrieval, and compare it to Spider-Agent [5] as an external baseline.

Exp. 1: Cryptic schemas break existing agents. We first confirm that obfuscation alone is enough to break a state-of-the-art agent. Since the clear and cryptic schemas differ only in their names, any drop must be caused by the obfuscation. As shown on the left side of Figure 4, when external knowledge documents are missing, Spider-Agent solves 41.6% of the clear questions but plummets to only 10.9% on the cryptic ones. By contrast, DoubleAgent is much more resilient and drops only from 51.5% to 43.6%. This gap persists even when external knowledge is available (Figure 4, right), where Spider-Agent solves 53.6% of the clear questions but only 29.3% of the cryptic ones, a collapse of 24.3 percentage points. The abbreviated names give it no signal to link a question to the right tables and columns, and it does not recover from the resulting failures.

Exp. 2: The knowledge graph closes most of the gap. Even without the knowledge graph, DoubleAgent already reaches 40.9% on cryptic schemas, well above Spider-Agent, since it retries failed queries to deduce names by trial and error. Adding the knowledge graph removes much of this guesswork and raises cryptic accuracy to 55.1%, cutting the obfuscation penalty from 13.5 to 5.3 percentage points. On clear schemas the gain is 6.0 percentage points, which confirms that the graph helps where the names are cryptic, not where they already carry meaning. An error analysis further shows that the knowledge graph nearly halves the number of errors due to schema-linking, from 83 to 44, so the dominant failure shifts from finding the right columns to writing correct SQL over them.

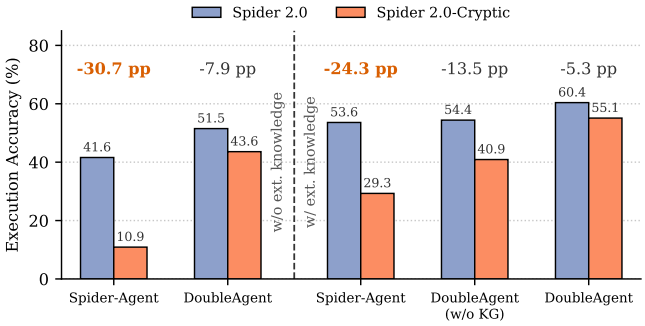

Figure 4: Execution Accuracy under Spider 2.0-Cryptic and Spider 2.0 ($n = 276$). Left: agents without external knowledge documents. Right: agents with access to external knowledge documents. The knowledge graph more than halves the accuracy drop that obfuscation causes.

Table 1: Tool calls per question on Spider 2.0-Cryptic (w/ ext. knowledge). DoubleAgent explores the knowledge graph instead of blindly probing the database.

Tool	Spider-Agent	DoubleAgent (w/o KG)	DoubleAgent
`query_kg`	-	-	**4.2**
`get_table_schema`	-	-	3.3
`execute_sql_query`	4.2	**10.6**	2.2
`preview_values`	-	1.3	0.7
`execute_bash`	**7.2**	-	-
Other KG tools	-	-	1.0
Total Avg. Calls / Q.	11.4	11.9	11.4

Furthermore, an analysis of the agent trajectories (Table 1) shows that while agents without a knowledge graph rely on blind database probing with `execute_sql_query`, the full DoubleAgent instead autonomously explores the enterprise knowledge graph (`query_kg`).

Exp. 3: The gain transfers to SAP S/4HANA. Finally, the same approach transfers beyond SQL to a real SAP S/4HANA system. On 516 OData questions across 11 business domains, the knowledge graph raises strict accuracy from 31.8% to 46.9% (Table 2). The gain is larger than on SQL, at ~15 against ~11 percentage points, since here the knowledge graph is crucial to route each question to the correct service among thousands. In addition, it slightly reduces the cost, from $4.92 to $4.22, since the agent wastes fewer steps in the

Table 2: SAP results ($n = 516$). The knowledge graph improves strict accuracy & lowers cost, at the price of higher latency.

	Strict Accuracy	Cost ($)	Runtime (s)
Baseline	31.8%	4.92	**0.81**
With KG	**46.9%**	**4.22**	2.45

exploration, although the knowledge graph-variant adds latency as the agent's SPARQL needs to be executed on the knowledge graph.

6 FUTURE DIRECTIONS

Spider 2.0-Cryptic and DoubleAgent are first steps towards Text-to-SQL on enterprise databases. While the absolute numbers of the observed effects may change with more experiments and newer models, we believe that we have identified clear gaps and directions for future work.

Completing the knowledge graph. Similar to many real-world enterprise knowledge graphs, our knowledge graph is incomplete. We infer keys from name patterns, generate descriptions, and add semantic edges, all of which are approximate. Therefore, a natural next step is a more elaborate approach to automatic knowledge graph completion, where missing keys, descriptions, and relationships are recovered more reliably than our heuristics allow.

Routing to the right data source. A preliminary error analysis on the SAP setting points to a more fundamental problem. The largest source of errors is not writing an incorrect query but selecting the wrong data source, since the agent must first route a question to the correct OData service among thousands before it can answer it. This is no longer a single-database problem and instead requires finding and combining the right sources, which is exactly the setting of our work on autonomous data agent collaboration, where a coordinator routes a question to the agents that can answer it [3]. We therefore intend to extend DoubleAgent in this direction.

ACKNOWLEDGMENTS

This research was funded by SAP, the German Federal Ministry of Research, Technology and Space (BMFTR) and the state of Hesse as part of the NHR program, as well as by the LOEWE Spitzenprofessur of the state of Hesse (III 5-519/05.00.003-(0005)), and by the German Research Foundation (DFG) under Germany's Excellence Strategy (EXC3057/1 "Reasonable Artificial Intelligence", Project No. 533677015). It was also partially funded by the BMFTR as part of the Software Campus research program (16IS23067) within the Project Management Agency for Artificial Intelligence. We also thank DFKI Darmstadt and hessian.AI for their support.

REFERENCES

[1] Jan-Micha Bodensohn, Ulf Brackmann, Liane Vogel, Anupam Sanghi, and Carsten Binnig. 2025. Unveiling Challenges for LLMs in Enterprise Data Engineering. *Proceedings of the VLDB Endowment* 19, 2 (Oct. 2025), 196–209. https://doi.org/10.14778/3773749.3773758

[2] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025. BEAVER: An Enterprise Benchmark for Text-to-SQL. arXiv:2409.02038 [cs.CL] https://arxiv.org/abs/2409.02038

[3] Timo Eckmann and Carsten Binnig. 2026. A Vision for Autonomous Data Agent Collaboration: From Query-by-Integration to Query-by-Collaboration. In *16th Annual Conference on Innovative Data Systems Research (CIDR '26)*. Chaminade, USA. https://www.vldb.org/cidrdb/papers/2026/p19-eckmann.pdf

[4] GraphQL Foundation. 2025. GraphQL Specification. https://spec.graphql.org/September2025/. Accessed: 2026-06-19.

[5] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763 [cs.CL] https://arxiv.org/abs/2411.07763

[6] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL] https://arxiv.org/abs/2305.03111

[7] Isaiah Onando Mulang', Felix Sasaki, Tassilo Klein, Jonas Kolk, Nikolay Grechanov, and Johannes Hoffart. 2025. SALT-KG: A Benchmark for Semantics-Aware Learning on Enterprise Tables. In *EurIPS 2025 Workshop: AI for Tabular Data*. https://openreview.net/forum?id=9vVMSvilGX

[8] Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards Robustness of Text-to-SQL Models Against Natural and Realistic Adversarial Table Perturbation. arXiv:2212.09994 [cs.CL] https://arxiv.org/abs/2212.09994

[9] Bhaskarjit Sarmah, Beez Siddighi, Anil K Rajput, and Akshay Valu. 2024. HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction. arXiv:2408.04948 [cs.IR] https://arxiv.org/abs/2408.04948

[10] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. arXiv:2002.10957 [cs.CL] https://arxiv.org/abs/2002.10957

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[12] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL] https://arxiv.org/abs/1809.08887