Revolutionizing Database Q&A with Large Language Models: Comprehensive Benchmark and Evaluation

Yihang Zheng*
yihang@stu.xmu.edu.cn
Xiamen University
Xiamen, China

Bo Li*
leebo@stu.xmu.edu.cn
Xiamen University
Xiamen, China

Zhenghao Lin
zhenghaolin@stu.xmu.edu.cn
Xiamen University
Xiamen, China

Yi Luo
luoyi@stu.xmu.edu.cn
Xiamen University
Xiamen, China

Xuanhe Zhou
zhouxh@cs.sjtu.edu.cn
Shanghai Jiaotong University
Shanghai, China

Chen Lin†
chenlin@xmu.edu.cn
Xiamen University
Xiamen, China

Guoliang Li
liguoliang@tsinghua.edu.cn
Tsinghua University
Beijing, China

Jinsong Su
jssu@xmu.edu.cn
Xiamen University
Xiamen, China

Abstract

The development of Large Language Models (LLMs) has revolutionized QA across various industries, including the database domain. However, there lacks a thorough evaluation regarding the capabilities of different LLMs in database QA. To this end, we introduce *DQABench*, the first comprehensive database QA benchmark for LLMs. *DQABench* features an innovative LLM-based method to automate the generation, cleaning, and rewriting of evaluation dataset, resulting in over 200,000 QA pairs in English and Chinese. These QA pairs cover a wide range of database-specific knowledge extracted from manuals, online communities, and DB instances, allowing for assessment of LLMs' Retrieval-Augmented Generation (RAG) and Tool Invocation Generation (TIG) capabilities in the database QA task. Furthermore, we propose a highly modular and scalable testbed *DQATestbed*, with basic and advanced components such as Fine-tuning, Question Classification Routing (QCR), RAG, TIG, and Prompt Template Engineering (PTE). Finally, we provide an evaluation pipeline that computes various metrics throughout a standardized evaluation process to ensure the accuracy and fairness. Our evaluation reveals the strengths and limitations of nine open-source and commercial LLMs, and the impact of various service components (e.g., fine-tuning, QCR, RAG, TIG). The proposed benchmark dataset is available at https://github.com/XMUDM/DQABench.

*Both authors contributed equally to this research.
†Corresponding author.

KDD '25, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3737405

CCS Concepts

• **Information systems** → **Information systems applications**.

Keywords

AI4DB; LLM; Database QA; Benchmark

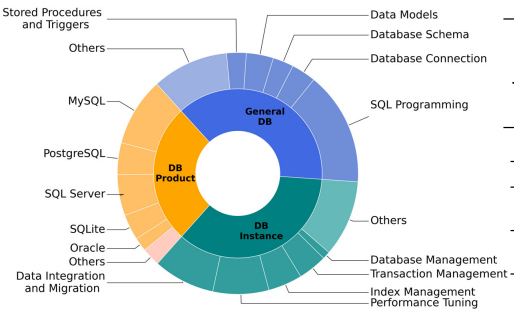
ACM Reference Format:
Yihang Zheng, Bo Li, Zhenghao Lin, Yi Luo, Xuanhe Zhou, Chen Lin, Guoliang Li, and Jinsong Su. 2025. Revolutionizing Database Q&A with Large Language Models: Comprehensive Benchmark and Evaluation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3737405

1 Introduction

Large language models (LLMs) have emerged as one of the most promising artificial intelligence technologies in recent years. LLMs have achieved remarkable progress in answering diverse questions from vast domains, such as medicine [55, 58], finance [22], earth science [6], law [9], and so on.

Question-answering (QA) systems are pivotal copilots for database systems. Traditionally, extensive IT knowledge, such as SQL, storage, hardware, and network, is necessary for database deployment and usage. This demand for expertise highlights the necessity for Database Question Answering (DBQA), which allows DB users to pose questions freely and receive technical support whenever and wherever needed.

People expect LLMs to excel in DBQA, yet **their potential remains under-explored due to the lack of high-quality benchmarks**. Existing QA benchmarks fall short in two key aspects: *(1) DB Expertise:* As shown in Figure 1(a), database expertise encompasses a broad and diverse range of categories, including general, product-specific (e.g., deployment and execution), as well as instance-level knowledge (e.g., monitoring, diagnostics, and optimization). While some existing QA benchmarks [16, 17] contain IT-related questions, their coverage for DBQA is limited and may exclude advanced or modern concepts such as "serverless" or "object



(a)

	Common Bench	RAG Bench	NL2SQL Bench	Agent Bench	Ours Bench
DB Knowledge	A few topics	×	SQL	×	Full Coverage
Retrieval	×	Most News/Wiki	×	×	Professional DB Docs
Tool Invocation	×	×	SQL Execute	Common Tools	Comprehensive DB Tools
Planning	×	×*	×	×*	Multi-turn Interaction

(b)

Question Type	Vision Bot Answer	GPT-4 Answer
General DB What does the Severless mean ?	"Serverless" refers to a cloud computing model …	"Serverless" refers to a cloud computing model … ✓
Product-specific Does PG support JSON_TABLE ?	No.17 version (newest) of PG docs show that PG support ...	PG does NOT support JSON_TABLE ✗ Wrong Answer !
Instance-specific Why do SQL queries execute slow ?	Query_Analyasis Tool shows that query need to be rewritten...	The reason may: (1) Query (2)CPU (3) IO… ✗ No Targeted !

(c)

Figure 1: (a) Percentage of various questions in online DB communities. (b) Existing benchmarks comparison. *Note: x* means a few bench have limited implementation* (c) GPT-4 DBQA examples.

storage." *(2) Multi-Dimensional Evaluation:* DBQA requires assessing LLMs across various capabilities, including auxiliary information usage, tool application, reasoning, etc. However, as shown in Figure 1 (b), most of the benchmarks focus on only a few dimensions and often overlook the core needs of DBQA. For instance, most QA benchmarks [27] focus only on a standalone LLM, RAG benchmarks [17, 21, 30] mainly evaluate retrieval from news or Wikipedia. In contrast, database documentation poses a more complex retrieval challenge with its long contexts, technical terms, and structured code. Likewise, Text2SQL benchmarks [11, 23, 54] focus solely on SQL generation, and LLM-based agent benchmarks [5, 37, 47, 48] only test basic tool use. In contrast, DBQA places a deeper demand on multi-turn interactions and tool-planning capabilities.

Furthermore, existing LLMs currently perform poorly in DBQA. As shown in Figure 1(c), even GPT-4 struggles with advanced concepts from technical documentation and tool-invocation DBQA. Our experiments confirm that, even with RAG and tool-calling modules, most LLMs can only correctly answer less than 40% of simple DBQA questions.

Therefore, **it is essential to propose a comprehensive DBQA benchmark** to advance both the efficiency of DBMS and the development of LLMs. However, such evaluation faces three major challenges: (1) *DBQA Dataset*, as data collected from the Web [15, 17, 35, 36] often exhibits low question quality, factually incorrect or subjective answers, and limited diversity; (2) *DBQA Testbed*, since a complete DBQA system must integrate multiple components (e.g., pre-training, fine-tuning, routing, retrieving, and tool invocation); and (3) *Multi-Dimensional DBQA Evaluation*, DBQA need various fine-grained metrics and assessment of both intermediate components and end-to-end performance.

We make the following contributions.

(1) *To address C1*, we propose the first benchmark ***DQABench*** to evaluate question-answering performance in the database domain. ***DQABench*** simulates real-world DB scenarios, covers a wide range of DB topics, and contains complex questions that demand assistance from external manuals and DB tools. The dataset contains 200,000 QA pairs, larger than existing instruction datasets in the IT field [27]. We propose novel strategies to enrich online resources for high-quality questions, answers, and annotations. (Section 3)

(2) *To address C2*, we propose a plug-and-play testbed to experiment with different LLM application strategies. The testbed assembles all components potentially involved in DBQA, such as Question-Category Routing (QCR), Prompt-Template Engineering (PTE), Retrieval-Augmented Generation (RAG), and Tool-Invocation Generation (TIG). (Section 4)

(3) *To address C3*, we conduct an in-depth evaluation of the end-to-end performance of nine LLMs (Section 5) and their modular performance using various components. We discover several insights, including but not limited to the following four key aspects. (Section 6)

I1: Necessity of DBQA Research. We find that even the most powerful GPT-4 achieves less than 60% accuracy on the simple questions in ***DQABench***, underscoring the need for further DBQA research.

I2: End-to-End DBQA Ability. We observe significant performance variations across different LLMs. Smaller models, in particular, struggle with advanced questions that require tool usage and extensive knowledge sources.

I3: Impact of Intermediate Components. We showcase that a complete pipeline with various intermediate components (such as pre-training, fine-tuning, QCR, RAG, and TIG) is important to DBQA performance.

I4: Development Directions for Intermediate Components. We compare various current implementations of intermediate components to identify performance bottlenecks and future directions. For example, the QCR module benefits from a well-trained classifier rather than relying on LLM responses, and the improvement from the RAG module is more constrained by recall rate rather than the interaction method between the LLM and the retrieval module, etc.

2 Related Work

Researchers have shown that scaling model size and training data improves question-answering performance. Large-scale LLMs such as GPT-3.5 [31], LLaMa [42], and PaLM [1] demonstrate capabilities far beyond traditional models—with GPT-4 [32] achieving human-level performance on many benchmarks. Medium- and small-scale models (e.g., Llama3-7B [42], Mistral-7B [19], Baichuan2-13B [51],

Table 1: *DQABench* dataset statistics

Type	Source	# Q.	English		Chinese		Label
			|Q|	|A|	|Q|	|A|	
General	Exam	2,152	224.19	458.61	85.44	124.59	N/A
	Forum	74,893	1205.75	1273.16	354.15	790.24	
Product Specific	OpenGauss	21,689	78.19	666.26	31.34	267.42	Retrieval Label
	GaussDB	4,950	84.84	885.67	34.62	394.60	
Instance Specific	Common	1,080	86.07	1070.21	29.76	757.23	Tool Label
	Expand	2,707	127.18	1019.28	34.81	515.92	

Qwen-14B [3], and Yuan-2B [46]) have been developed for deployment on resource-constrained devices, though their performance is typically limited to simpler tasks. Moreover, external knowledge has been incorporated to further enhance answer quality by retrieving documents [12] or guidelines [28], leveraging structured knowledge bases (e.g., knowledge graphs) for reliable reasoning [40], or using LLMs as agents to invoke external tools [5, 44, 47, 48].

In vertical domains such as medicine [55, 58], finance [22], earth science [6] and law [9], specialized LLMs are created by performing domain-specific pre-training and fine-tuning. This process adapts a general-purpose model to capture the unique language styles, terminologies, and deep expertise of a field. For example, in medicine [25, 55, 56, 58] the models integrate clinical terminology and evidence-based guidelines, while in finance [22, 49] they incorporate industry jargon and regulatory context; similarly, in earth science [6] and law [9], models are fine-tuned with data that reflect complex scientific measurements or legal reasoning. Various benchmarks are proposed to evaluate LLM performances in vertical domains [13, 14, 25, 49, 56].

In the database domain, LLMs have been shown to ground database tuples, schemas, and queries in novel ways [10], and some proposals even suggest they could serve as Database Administrators [60]. Other studies use LLMs for intelligent database diagnosis (e.g., D-Bot [61]) and query rewriting [26]. LLMs also perform strongly in converting natural language to SQL (NL2SQL); systems such as Binder-SQL [8], DIN-SQL [33], and BIRD [24] generate SQL directly from natural language, while DB-GPT [50] offers complete visualized data analyses and reports. Established benchmarks [11, 23] further support the evaluation of these capabilities. Notably, our work goes beyond NL2SQL by considering tool invocation and ensuring that LLM outputs meet varied formatting requirements.

3 *DQABench* Dataset Generation

A dataset consisting of pairs of questions and answers is crucial for evaluating the performance of a DataBase Question-Answering (DBQA) bot. However, manually creating such pairs is labor-intensive. This section introduces techniques for generating a dataset tailored to the DBQA benchmark.

As shown in Figure 1 (a), the DB questions can be divided into three subsets (general DB, product-specific and instance-specific questions), corresponding to three key skill sets of an LLM-based DBQA bot. The three question categories are different in (1) data sources: general DB questions are publicly available, while product-specific questions and instance-specific questions are almost impossible to obtain complete examples online. (2) problem background: the latter two categories (i.e., product-specific questions and instance-specific questions) need supporting information, such as the product manual and instance context. (3) ground-truth answers: the latter two categories must provide retrieval results or tool invocation results to demonstrate the reasoning and produce trustworthy answers.

Accordingly, we propose methods to construct the dataset for each category. (1) We tailor a prompt-chain for each DBQA subset based on its characteristics, ensuring that generated QA pairs cover a wide range of topics, present clearly defined questions, and yield factually accurate answers. (2) We leverage LLMs to achieve alignment of the answer's language style with a tone that is user-friendly, logically clear, and specific, rewriting any answers that do not meet this alignment—thereby enhancing the dataset's training value. (3) For each subset, we develop methods to extract complete supporting information for every QA pair, ensuring the dataset provides the extra details necessary for multidimensional evaluation.

As shown in Table 1, we construct a dataset with bi-lingual QA pairs on the three categories, translating English pairs into Chinese and vice-versa, leading to a total of over 200,000 QA pairs. According to our statistics, our QA covers, but is not limited to, the following DB domains: Operation Diagnostics, Business Intelligence, Performance Monitoring and Tuning, Data Analysis, System Utilization Analysis, Deployment Issues, Operations Management, question Optimization, Backup and Recovery, Permission Management, Index Management, and Database Tuning.

3.1 General DB QA

Data Sources. We have two types of data sources. (1) Similar to other domain-specific datasets, we collect 2,000 unique multiple-choice questions from four DB university textbooks, three online courses, and 28 online course exams. (2) To ensure that the ***DQABench*** dataset covers a wide range of questions asked by DB users in daily usage, we collect QA entries from the largest English and Chinese online DB communities.

Step 1: Question Filtering.We filter the collected content based on online feedback. First, we compute the ROUGE-1 score, which measures the overlap of unigrams between questions and then QA with a large ROUGE-1 score (≥ 0.8) are merged. For each question, we retain only the accepted answers and those with high upvotes (≥ 50) to ensure the factual correctness of the responses.

Step 2: Answer Rewriting. The collected answers are insufficient as ground truth. For instance, exam questions only offer letter options, leading LLMs to generate overly random responses. To address this, we prompt GPT-4 to add detailed explanations for each exam answer choice. We instruct GPT-4 to rephrase each accepted online response into a detailed, professional, and friendly style, resulting in more specific and user-friendly content.

3.2 Product-Specific QA

Constructing product-specific QA pairs from online sources is challenging because it's hard to verify if online answers are based on specific product documentation or to locate that documentation for evaluation. Therefore, we build these QA pairs using the workflow in Figure 2.

Step 1: Pre-processing Manuals. Most product manuals are too long for LLMs to handle. We segment each manual into parts that contain complete paragraphs and do not exceed 8,000 tokens.

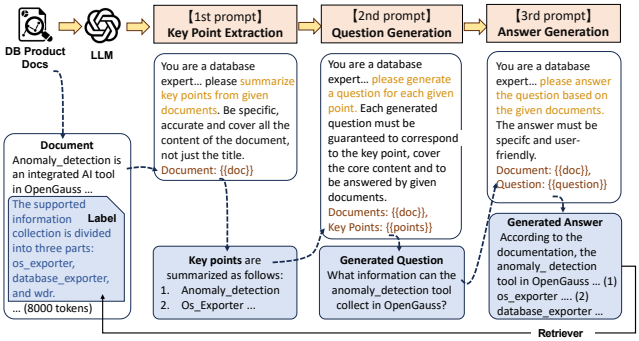


Figure 2: Generation of product-specific QA

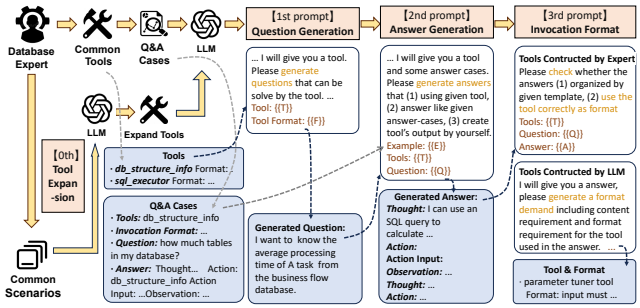


Figure 3: Generation of instance-specific QA

This allows LLMs to process the documents more precisely and generate more detailed QA pairs.

Step 2: QA Generating. To reduce manual efforts, we use LLMs to generate several QA pairs on each document segment. The challenge is, directly instructing the LLM to generate QA results in low-quality outcomes. Specifically, the generated questions can overly focus on minor details while neglecting the main points of the given document segment, the answers can be too concise, and the QA pairs can be repetitive, lacking a diverse coverage of possible topics. Thus, we propose a novel prompt chain to generate QA. As shown in Figure 2, the prompt chain first requires LLMs to summarize the document segment's key points. Then, the prompt chain demands LLMs to generate a question for each key point that can be answered based on the document segment. Finally, the prompt chain asks LLMs to produce a detailed, user-friendly answer.

Step 3: Retrieval Label Annotating. Since the dataset evaluates the QA bot's ability to apply external knowledge and adapt to different DB products in RAG scenarios, we also annotate relevant text chunks. We use the generated question and answer as a query to identify the fine-grained chunks with relevant information (with cosine similarity ≥ 0.8).

3.3 Instance-Specific QA

It is infeasible to construct instance-specific QA pairs from online sources. Online questions are almost always incomplete due to privacy reasons, missing necessary instance-level contextual information, such as the database's table structures, workload information, etc. Therefore, we have to generate instance-specific QA pairs by LLMs automatically.

There are numerous database *tools* provided in DBMS that support database monitoring, optimizing, and analyzing for DB developers, administrators, and analysts. The LLM's proficiency in answering instance-specific questions relies on whether LLMs can accurately invoke different tools to obtain the instance's contextual information. Thus, as shown in Figure 3, our dataset construction workflow starts with building a pool of DB tools.

Step 1: Constructing DB Tool Pool. (1) We first survey the DB tools commonly used in real-production systems. We identify six types of *common DB tools* that are frequently used for data modeling, database monitoring and performance optimization (Details in the Appendix 8). The implementation of each tool type, including the exact tool name and the format of input and output, may vary for different DBMS products. (2) The common tools can not cover all DB tools, especially new ones developed to meet the demands of a real-production system. We expand the common tools to a set of *generalization tools* to evaluate the QA bot's generalization ability to utilize different DB tools properly. We determine DB scenarios and require GPT-4 to generate imaginary DB tools that can benefit these DB scenarios.

Step 2: Generating Questions. For each tool, we require GPT-4 to generate questions that can be solved by the target tool using the prompt in Figure 3. Moreover, we want to effectively evaluate the QA bot's *planning* ability, which involves adequately combining several DB tools and organizing tools with the right action order. Thus, we manually construct three to four questions for each common tool and scenario that demand a chain of multiple tool invocations, and we use these questions as few-shot examples to encourage GPT-4 to generate complex questions.

Step 3: Generating Answers. (1) First, we manually produce *answer cases* for manually constructed questions above. The DB experts compose the answer cases in the following procedure: construct a real DB instance according to the description in the question, call the DB tools when necessary, and answer the question based on real tool feedback. (2) Then we use the answer cases as few-shot learning examples to guide GPT-4 to generate answers efficiently. We adopt the Chain-Of-Thought(COT) prompting technique to generate an answer for each question following ReACT [53].

Step 4: Polishing Answers. Finally, we ask GPT-4 to rethink and polish the answer. This step is different for common tools and generalization tools. (1) For each answer to common tools, since the common tools are real DB tools with pre-defined formats of tool output, we ask GPT-4 to examine its output to ensure the answer format is correct to trigger the tool. (2) For each answer relating to generalization tools, since the generalization tools are imagined and reasonably inferred by GPT-4, they do not have a pre-defined format; we ask GPT-4 to summarize the tool's format.

4 *DQATestbed*

When adopting a general-purpose LLM for DB QA, various auxiliary modules are indispensable to leverage and adapt the LLM's general knowledge of linguistic patterns and common senses into the DB environment. Currently, there lacks an all-encompassing DBQA testbed that incorporates LLM and various auxiliary modules.

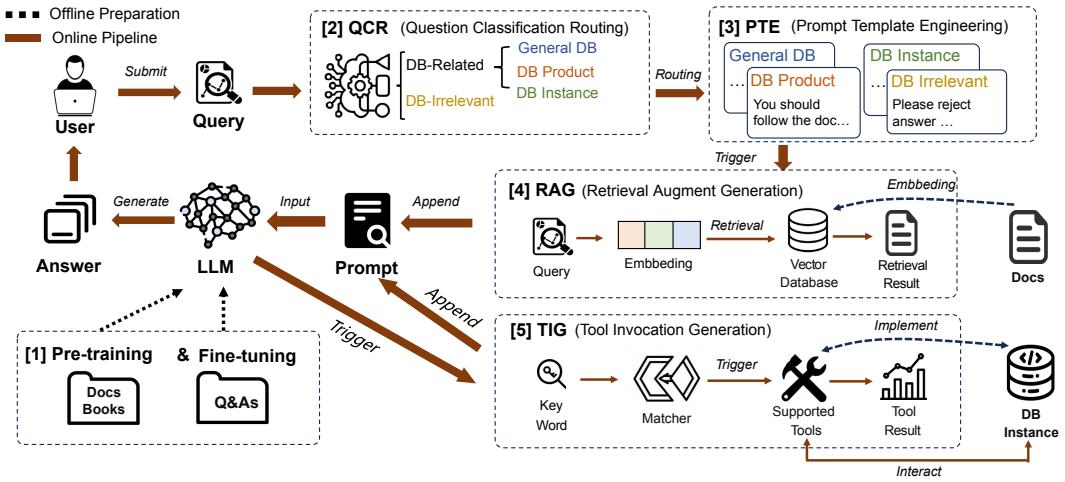


Figure 4: The Overview of *DQATestbed* Framework.

Table 2: Comparison of LLM-based DBQA Solutions.

Solution	Pre-training	Fine-tuning	Question Routing	Retrieval Augment	Tool/ Agent
LLM-only	✓	✓	X	X	X
Langchain [7]	X	X	X	✓	X*
D-bot [62]	X	X	X	✓	X★
DB-GPT [50]	X	✓	X	✓	X★
Ours	✓	✓	✓	✓	✓

Note: ✓: fully supports the component. X: lacks functionality completely. X*: DB tools need to be customized in the Langchain framework. X★: limited support, D-bot focuses on data interaction issues, and DB-GPT focuses on database operational diagnosis.

Table 2 compares the completeness of the proposed testbed with recent LLM adaptation frameworks. Existing works overlook some important modules. On the contrary, our proposed testbed supports a full chain of auxiliary modules for LLM's domain adaptation. We believe that these modules represent a future design paradigm for DBQA systems. We will describe and analyze the contribution of each module to DBQA performance, demonstrating their necessity in well-designed LLM-based DBQA systems of the future.

The workflow of the proposed testbed is shown in Figure 4.

Offline. Before deployment, the core LLM module goes through the stage of *continual pre-training and fine-tuning* to acquire more specific DB concepts and skills while preserving the LLM's general linguistic knowledge. The user can also load a knowledge source of documents (usually up-to-date materials that do not appear in the training corpus, or in-house data for privacy reasons) stored in the form of a vectorized database.

Online. When the user submits a query, it first goes through the *Question Classification Routing* (QCR) module to determine the logical structure of reasoning an answer. Depending on the result of QCR, i.e., the type of question, it is directed to an appropriate prompt in the *Prompt Template Engineering* (PTE) module. Keywords in the prompt generated by the PTE module will trigger the *Retrieval Augmented Generation* (RAG) or *Tool Invocation Generation* (TIG) module to append to the content of the prompt. For example, if the query is related to a certain DB product, then to mitigate hallucination, the RAG module is triggered to retrieve trusted data and generate more accurate and relevant answers. The process can iterate for a few rounds if needed. For example, if answering the query needs to perform data analysis, the schema tool is first triggered to fetch the table structure of the database. And then a selection tool is triggered to execute a SQL selection query, to compute the required data statistics in the database. Finally, the LLM is instructed by the prompt to generate the answer.

The specific components of the testbed are as follows.

Pre-training and Fine-tuning. Prior to deployment, the LLM undergoes continual pre-training and fine-tuning to acquire domain-specific database concepts while retaining its general linguistic capabilities. In addition, users may load a vectorized knowledge base containing up-to-date or private documents.

Question Classification Routing (QCR). This module automatically categorizes incoming queries and routes them to customized prompt templates, thus mitigating security risks and enhancing response accuracy. We explore three approaches: an LLM-based classifier, an XLNet-based classifier trained on annotated data, and a hierarchical classifier through multi-step classification.

Prompt Template Engineering (PTE). Customized prompt templates are designed for various query categories. Each template features dynamic slots (indicated by "{}") that are populated by auxiliary modules. For example, specific keywords trigger either the RAG module for DB product queries or the module for instance-specific queries.

Retrieval Augmented Generation (RAG). To enrich the LLM's responses with external knowledge, this module first segments documents from the knowledge base into text blocks, converts them into dense vectors, and stores them in a vector database. A submitted query is similarly vectorized and matched against these blocks. The retrieved content is appended to the prompt, guiding the LLM to generate more precise and context-relevant answers.

Six typical RAG techniques (1) Naive RAG [7] (2) RRR [29] (3) Iter-Retgen [38] (4) Self-Ask [34] (5) Active RAG [20] (6) Self-RAG [2] are evaluated under the RAG-Lab [59] framework.

Tool Invocation Generation (TIG). Leveraging a Chain-of-Thought (COT) prompt template [53], the TIG module enables the LLM to iteratively decide which database-specific tools to invoke. The process involves the LLM outputting a "Thought" to reason about the necessary tool, followed by an "Action" and "Action Input" that specify the tool and its parameters. After the tool executes and returns an "Observation," the LLM may invoke additional tools until sufficient information is gathered for a final answer.

5 End-to-End Performance Experiment

We examine the end-to-end performance of different LLMs, i.e., do they produce high-quality answers for database questions.

LLMs. Seven popular commercial and open-source LLMs are compared, including **(1) GPT-4 [32]**, the most powerful large-scale LLM currently released by OpenAI, using the GPT-4-0125-preview version. **(2) GPT-3.5 [31]**, a popular large-scale LLM currently released by OpenAI, using the GPT-3.5-turbo-0125 version. **(3) GLM-3 [57]**, a popular large-scale LLM for both Chinese and English, released by Zhipu AI. **(4) Llama3-8B [42]**, the latest mid-sized open-source LLM released by Meta AI, claimed to achieve SOTA performance among mid-sized models, using the Llama3-8B-Instruct version. **(5) Llama2-13B [43]**, the most popular mid-sized open-source LLM released by Meta AI, using the Llama1-13B-Chat version. **(6) Yuan2-2B [46]**, a popular small-sized open-source model for both Chinese and English, released by IEIT Systems, using the Yuan2-2B-Februa version. **(7) Baichuan2-13B [51]**, a popular mid-sized open-source model for both Chinese and English, released by Baichuan Intelligence, using the Baichuan2-13B-Chat version.

In addition to evaluating the performance of vanilla LLMs, we assess the impact of continued pre-training and fine-tuning. Limited by training cost, we only apply them upon Baichuan2-13B, leading to two additional LLM variants: **(8) Baichuan2-13B-sft**, which is a Baichuan2-13B variant fine-tuned. **(9) Baichuan2-13B-cpt-sft**, which is a Baichuan2-13B variant pre-trained and fine-tuned.

Evaluation pipeline. The workflow in Section 4, i.e., with auxiliary modules, can improve the answer quality of each tested LLM. Thus, this section presents the results generated by LLMs through the entire workflow implementing all modules of *DQATestbed*. Since each module can adopt various strategies that may introduce bias when comparing the performance between the LLMs, we use a standard pipeline with fixed intermediate output to obtain the best output of different LLMs.

5.1 Factual Accuracy on Simple Questions

We first extract simple QAs that involve only a single knowledge point and have clear true/false answers. We then use a GPT-4-based judging approach to determine whether the LLMs' answers met the factual correctness criterion. We also conduct evaluations by DB experts, and 96% GPT-4 judgments align with the experts' opinions, demonstrating the reliability of the GPT-4 judge.

Figure 3 shows that (1) current DBQA systems remain far from ideal. even the SOTA models such as GPT-4 achieve only around 50% accuracy on general and product-specific questions and 40% on instance-specific ones, indicating that current DBQA systems remain far from ideal; (2) models leveraging both pre-training (cpt) and fine-tuning (sft), like Baichuan2-cpt-sft, show clear improvements that underscore the importance of these techniques; (3) smaller models struggle with advanced questions that require tool usage and extensive knowledge sources.

Table 3: Factual accuracy for simple EN DBQAs

Models	General DB	Product Specific	Instance Specific
GPT-4	0.51	0.50	0.42
GPT-3.5-turbo	0.31	0.42	0.28
GLM-3-turbo	0.33	0.48	0.18
Llama3-8B-Instruct	0.32	0.44	0.20
Llama2-13B-Chat	0.04	0.15	0
Yuan-2B	0.01	0.06	0
Baichuan2-13B-Chat	0.14	0.23	0.05
Baichuan2-sft	0.23	0.46	0.52
Baichuan2-cpt-sft	0.29	0.58	0.57

5.2 Comprehensive Evaluation

The DBQAs often require comprehensive solutions, complex code design, detailed conceptual explanations, and multi-faceted proposal analyses. These types of QAs are not easily reduced to a simple binary factual accuracy judgment. Beyond accuracy, factors like logical coherence, specificity, and user-friendliness of the language style are also critical in evaluating an LLM's capability. To capture these broader dimensions, we adopt the widely used WinRate metric from the NLP community. In the WinRate judging prompt, we stipulated that factual error would result in a loss, while overall judgment also considers the answer's logical structure, depth of detail, illustrative examples, and user-friendliness. This holistic evaluation offers a more nuanced comparison of LLM performance in the DBQA domain.

We compute the WinRate of each LLM versus two baselines. *(1) GPT-3.5-Turbo (Vanilla)*: we input the user's question to GPT-3.5-Turbo without employing any prompts. This result demonstrates the difference between a dedicated DBQA system (i.e., the testbed) and typical LLM applications. *(2) GPT-3.5-Turbo (Testbed)*: we also equip GPT-3.5-Turbo on the testbed. This result emphasizes the inherent capability difference of each LLM. We have the following insights from Table 4.

The experiment result shows that (1) larger models and richer pre-training data boost DBQA performance—GPT-4 sets the state-of-the-art on general DB questions, and Llama3-8B, leveraging 15T tokens, gains significant improvements on the ***DQABench*** dataset; (2) domain-specific continual pre-training and fine-tuning enhance DBQA, with Baichuan2-13B's continual pre-training raising scores by 0.28–0.39 (80%–111%) and fine-tuning adding another 0.28–0.38 (80%–136%), even allowing Baichuan2-13B-sft to outperform GPT-4 on instance-specific questions; (3) RAG and TIG strategies further benefit LLMs in DBQA tasks, as indicated by lower WinRate against GPT-3.5-Turbo testbed; and (4) smaller LLMs struggle with DB tool usage, since instance-specific inputs and outputs demand robust

Table 4: WinRate of different LLMs versus the competitor

Model	$\|\theta\|$	DB General		Product-specific		Instance-specific		Average	
		ZH	EN	ZH	EN	ZH	EN	ZH	EN
WinRate v.s. Vanilla GPT-3.5-Turbo									
GPT-4		0.85	0.95	0.86	0.86	0.69	0.53	**0.80**	**0.78**
GPT-3.5-Turbo		0.53	0.56	0.60	0.60	0.58	0.57	**0.57**	**0.58**
GLM-3-Turbo		0.63	0.62	0.81	0.58	0.44	0.44	**0.63**	**0.55**
Llama3	8B	0.60	0.67	0.79	0.75	0.37	0.40	**0.59**	**0.61**
Llama2	13B	0.12	0.09	0.35	0.41	0	0	**0.16**	**0.17**
Yuan2	2B	0.03	0.02	0.22	0.18	0	0	**0.08**	**0.07**
Baichuan2(vanilla)	13B	0.27	0.29	0.56	0.60	0.23	0.15	**0.35**	**0.35**
Baichuan2-sft	13B	0.44	0.31	0.88	0.76	0.90	0.82	**0.74**	**0.63**
Imp. w.r.t. vanilla		+0.17	+0.02	+0.32	+0.16	+0.67	+0.67	+0.39	+0.28
Baichuan2-cpt-sft	13B	0.57	0.48	0.88	0.74	0.91	0.87	**0.79**	**0.70**
Imp. w.r.t. vanilla		+0.30	+0.19	+0.32	+0.14	+0.68	+0.72	+0.44	+0.35
WinRate v.s. GPT-3.5-Turbo (Testbed)									
GPT-4		0.83	0.95	0.64	0.68	0.90	0.64	**0.79**	**0.76**
GPT-3.5-Turbo		-	-	-	-	-	-	-	-
GLM-3-Turbo		0.62	0.65	0.66	0.57	0.55	0.49	**0.61**	**0.57**
Llama3	8B	0.60	0.65	0.62	0.51	0.49	0.52	**0.57**	**0.56**
Llama2	13B	0.12	0.06	0.36	0.16	0	0	**0.16**	**0.07**
Yuan2	2B	0.03	0.02	0.13	0.07	0	0	**0.05**	**0.03**
Baichuan2(vanilla)	13B	0.26	0.30	0.42	0.40	0.16	0.11	**0.28**	**0.27**
Baichuan2-sft	13B	0.44	0.30	0.68	0.66	0.85	0.87	**0.66**	**0.61**
Imp. w.r.t. vanilla		+0.18	0	+0.26	+0.26	+0.69	+0.76	+0.38	+0.34
Baichuan2-cpt-sft	13B	0.55	0.42	0.65	0.73	0.95	0.90	**0.72**	**0.68**
Imp. w.r.t. vanilla		+0.29	+0.12	+0.23	+0.33	+0.79	+0.79	+0.44	+0.41

Table 5: Multi-dimensional Evaluation of Different LLMs

Model	General			Product-Specific			Instance-Specific		
	GE	Pr	Ip	GE	Pr	Ip	GE	Pr	Ip
GPT-4	6.19	96.80	97.45	5.77	78.60	84.30	4.06	50.05	53.40
GPT-3.5-turbo	4.57	81.50	82.85	5.41	63.55	70.50	3.59	46.80	49.50
GLM-3-turbo	4.93	90.25	91.65	6.13	72.95	80.25	3.19	39.80	42.75
Llama3-8B-Instruct	4.73	90.00	91.90	5.95	68.65	76.00	3.33	49.20	44.50
Llama2-13B-Chat	1.36	21.60	24.60	4.69	47.65	44.65	0	0	0
Yuan-2B	1.44	21.20	25.40	2.51	27.20	27.85	0	0	0
Baichuan2-13B-Chat	3.47	75.25	80.05	4.29	54.55	61.65	0.48	6.40	16.50
Baichuan2-sft	3.73	69.95	65.90	5.63	74.65	79.55	5.67	69.50	70.85
Baichuan2-cpt-sft	4.16	80.10	82.55	6.64	85.40	88.10	5.87	73.70	73.35

instruction-following that models like Llama2 and Yuan2 lack, underscoring the need for targeted instruction tuning.

5.3 Multi-dimensional Evaluation

We conducted a multi-dimensional evaluation of LLMs' end-to-end DBQA performance, extending beyond traditional metrics like accuracy and win rate. The assessment framework, aligned with the accuracy evaluation datasets, comprises three key dimensions: **(1)GE (G-Eval):** Comprehensive scoring of LLM-generated answers. **(2)Pr (Practicality):** Assessment of answer's practical utility and actionable guidance. **(3)Ip (Interpretability):** Measurement of answer's conceptual clarity and logical transparency.

The experimental results, illustrated in Figure 5, reveal three critical findings: (1) **Dimensional Consistency:** LLMs demonstrate consistent performance patterns across all metrics, with no significant discrepancies between dimensions. (2) **RAG-Induced Degradation:** Advanced models like GPT-4 exhibit compromised usability and interpretability when processing RAG content, indicating the lack of LLM human-friendly alignment for holistic quality in RAG scenarios. (3) **Instance-Specific Limitations:** Challenges persist in maintaining usability and interpretability for questions

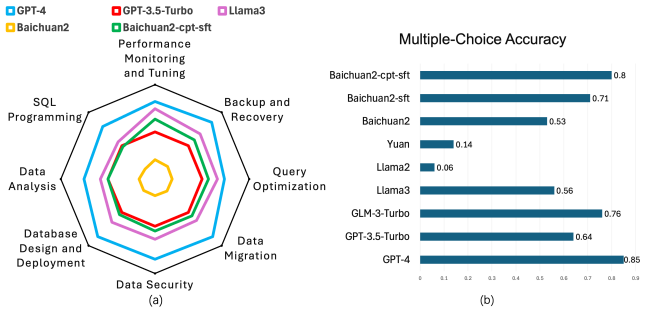


Figure 5: Performance on "DB General" questions

requiring instance-specific reasoning, underscoring the need for improved tool invocation logic to enhance overall response quality.

5.4 In-depth Analysis on DB General Questions

The DB-general questions require LLMs to depend on their inherent knowledge, with each question often demanding a distinct skill set. For example, the LLM needs to be proficient in SQL grammar to solve "Write a SQL to create index" and knowledge of index advisor to answer "which index is better." There are two types of questions in the subset, namely the subjective questions and the objective questions with multiple choices. In this section, we analyze the answers to DB-general questions from the two perspectives.

First, we report the WinRate of various LLMs in answering subjective questions. Specifically, we utilize GPT-4 to assign detailed labels to each question in the "DB general" subset, incorporating predefined few-shot labels from the prompt and allowing GPT-4 to generate new labels autonomously. We identify the eight most common labels: "Performance Monitoring and Tuning", "Backup and Recovery", "Query Optimization", "Data Migration", "Data Security", "Database Design and Deployment", "Data Analysis" and "SQL Programming". These labels cover 93.79% of the questions in the "DB general" subset.

The experimental results, illustrated in Figure 5 (a)'s radar chart, lead to the following conclusions: (1) Numerically, the response capabilities of different models in each sub-field correlate positively with their model size and overall ability, with no model being highly specialized in any particular field. (2) Regarding shape proportion, the radar charts of GPT-3.5 and GPT-4 are similar, showing balanced capabilities across the eight aspects. In contrast, Llama2, Llama3, Baichuan2, and other models based on the llama architecture display a similar pattern, excelling in Performance Monitoring and Tuning but weaker in SQL Programming. This indicates that GPT series models are better at generating accurate SQL Programming instructions, while llama-based models excel in comprehensive subjective analysis.

Next, we report the Multiple-Choice Accuracy, measured on the objective questions. As shown in Figure 5 (b), MCA aligns closely with the WinRate on subjective questions. This validates that questions in ***DQABench*** are set with an appropriate difficulty level and require a good understanding of DB knowledge to answer.

Table 6: WinRate (w/ Routing v.s. w/o Routing)

Models	WinRate V.S. Self w/o Routing			
	Product-Specific		Instance-Specific	
	ZH	EN	ZH	EN
GPT-4	0.85	0.75	0.69	0.61
GPT-3.5-turbo	0.73	0.76	0.59	0.60
GLM-3-turbo	0.82	0.76	0.44	0.53
Llama3-8B-Instruct	0.84	0.78	0.41	0.60
Llama2-13B-Chat	0.90	0.81	0	0
Yuan-2B	0.96	0.94	0	0
Baichuan2-13B-Chat	0.68	0.73	0.26	0.07
Baichuan2-sft	0.52	0.65	0.87	0.82
Baichuan2-cpt-sft	0.56	0.73	0.86	0.86

Table 7: Classification Performance

Method	F1-score					ACC	Latency
	Unsafe	General	Gauss	Tool	No-DB		
ZH							
GPT-4	0.77	0.48	0.47	0.25	0.59	0.55	2.64s
XLNet	0.95	0.89	0.92	0.91	0.79	0.90	0.39s
Hierarchical	0.95	0.91	0.98	0.99	0.87	0.94	0.68s
EN							
GPT-4	0.80	0.57	0.37	0.49	0.69	0.63	2.61s
XLNet	0.91	0.92	0.81	0.90	0.79	0.87	0.37s
Hierarchical	0.92	0.97	0.88	0.93	0.91	0.92	0.67s

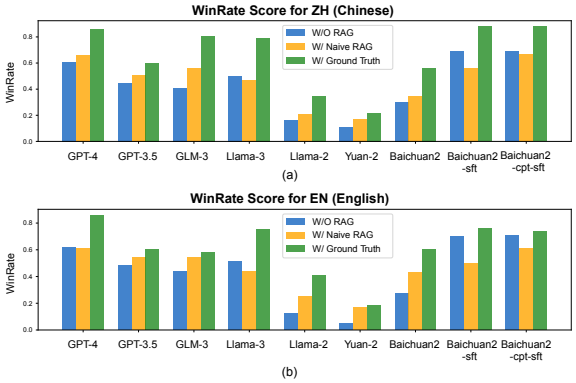


Figure 6: WinRate vs. GPT-3.5-Turbo (vanilla)

6 Modularized Evaluation

In this section, we rigorously assess the performance of each module of *DQATestbed*. We aim to achieve two goals: (1) verify the necessity of adopting each module for DBQA to improve answer quality; (2) evaluate the strengths and weaknesses of various solutions for each module in the context of DBQA.

6.1 Modularized Evaluation on QCR

We first demonstrate that using the QCR module to classify and route questions—thus enabling customized answers—is necessary. We implement two versions of each LLM to test whether question routing is necessary for DBQA systems. (1) *LLM w/ Routing*: using the QCR model for question classification and feeding the LLM with customized prompts according to the question type. We use the ground truth question type in ***DQABench***. (2) *LLM w/o Routing*: prompting the LLM using the "General DB" prompt template for all questions. Table 6 reports the WinRate of LLM w/Routing v.s. LLM w/o Routing on two question types, i.e., Product-Specific and Instance-Specific, because DB General questions are treated using the same prompt and thus receive the same result.

The results indicate that: (1) Most WinRate values exceed 0.5 (meaning LLM w/ Routing surpasses LLM w/o Routing), which suggests that utilizing query routing to customize responses based on question type significantly enhances the LLM's performance. Recent studies have demonstrated the importance of prompt engineering [45]. The QCR module can be seen as a dedicated, prompt engineering strategy tailored for DBQA that autonomously gives instructions and organizes examples. (2) In a few cases, particularly on instance-specific questions, the WinRate values fall below 0.5 or even reach zero. This is due to the model's limited ability to invoke tools: models lacking any tool invocation capability cannot provide effective customized responses, and thus, QCR can not improve their performance.

We then investigate which question classifier in Section 4 is effective. The testing data include DB-related questions from ***DQABench***, safe but DB-irrelevant questions from Alpaca [41] and Longbench [4], and unsafe labeled questions from Safety-Prompts [39] and Beaver-Tails Evaluation [18]. The F1 score for each category, accuracy, and response latency deployed on a workstation with an RTX-3090 GPU card are shown in Table 7.

From the experimental results, we have the following conclusions: (1) XLNet and Hierarchical are more accurate (+0.35) and faster (6.7x) than GPT-4. This is reasonable because general-purpose LLMs like GPT without fine-tuning perform worse on specific tasks than smaller models that are specifically trained. (2) There is a trade-off between latency and accuracy for small models. As shown in Table 7, the hierarchical classifier can achieve approximately a 0.05 performance improvement with 300ms more inference cost.

6.2 Modularized Evaluation on RAG

We have demonstrated the importance of RAG in end-to-end DBQA in Section 5 by the sharp performance rise with RAG. In Section 5, GPT-3.5-Turbo is provided the ground-truth retrieval text blocks. In this section, we want to investigate whether RAG with incorrect retrieval results can enhance answer generation.

First, we implement three versions of the nine LLMs to be evaluated. (1) w/o RAG, the LLMs are prompted to generate answers without the external knowledge provided by the RAG module; (2) w/ Naive RAG, the testbed retrieves the relevant text blocks by directly searching the vector database; (3) w/ Ground-truth RAG, the testbed uses the retrieval ground-truth from ***DQABench***. We report the WinRate score of each LLM version v.s. GPT-3.5-Turbo (vanilla) on the "product-specific" sub-dataset in Figure 6.

From the results, we have the following observations. (1) If the ground-truth retrieval texts are provided, the RAG module can significantly enhance the performance of general-purposed LLMs of any size, i.e., the seven LLMs. The performance improvements are more pronounced in smaller models, e.g., a 2.4× improvement on Llama2. This observation reveals the substantial value of RAG for edge-deployed models. However, the improvement is under ideal conditions, while the actual retrieval accuracy is not guaranteed in

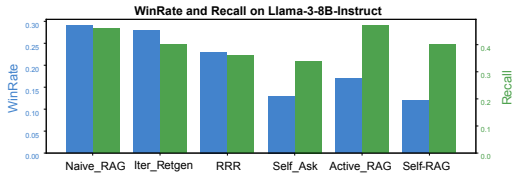


Figure 7: Performance of Different RAG Solutions

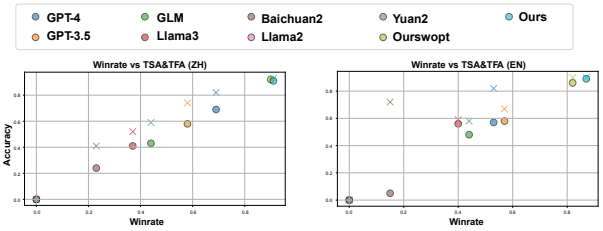


Figure 8: TFA and TSA vs. Performance of Different LLMs

real applications. (2) The ground-truth RAG implementation has brought minimal performance improvement for models fine-tuned with domain knowledge from databases, i.e., the last two LLMs Baichuan2-sft, and Baichuan2-cpt-sft. This observation suggests a significant overlap between the improvements brought by model fine-tuning and those from the RAG module, indicating that deploying either technique alone is sufficient within a limited budget. (3) With the naive RAG implementation, the performance increase reasonably shrinks for most LLMs. Furthermore, the performance improvement is negligible for GPT-4, and we observe a performance decline on Llama-3, Baichuan2-sft, and Baichuan2-cpt-sft. Detailed case studies indicate that this is primarily due to low recall rates. LLMs generate incorrect responses by grounding on irrelevant documents, ultimately compromising their ability to produce high-quality answers for questions they could have answered otherwise correctly without RAG.

Second, we investigate the impact of various RAG techniques implemented in *DQATestbed*. We report two metrics in Figure 7: (1) *Recall rate*: the average number of relevant text blocks successfully identified in the top-3 retrieval results, where the ground-truth labels contain one relevant block for each question. Since the result may not be completely identical with the ground-truth text block, i.e., they have overlapping parts. To determine whether a result is relevant, we compute the ROUGE-5, i.e., the overlap of sequences of five consecutive characters between the result and the ground-truth. If `ROUGE-5 > 0.15`, we determine a result is relevant. (2) *WinRate* calculates the performance of Llama3-8B-Instruct with the aforementioned RAG methods, compared with GPT-3.5-Turbo.

We have the following observations. (1) All existing RAG techniques yield low recall rates (below 50%), highlighting the main challenge in DBQA: accurately locating relevant documents. (2) For RAG methods that generate a passage based on the retrieval results, higher retrieval performance generally leads to higher answer quality, i.e., the recall rate positively correlates with the WinRate on Naive_RAG, Iter_Retgen, and RRR. (3) Techniques using a sentence-by-sentence generation strategy, such as Self-Ask, Active RAG, and Self-RAG, harm the QA performance. Because these RAG strategies divide the answer passage into sentences and generate sentences based on relevant documents at the sentence level, their retrieval module has been called more often, which raises the risk of encountering irrelevant information. For example, we identify an erroneous case where the question concerns only the `log_dir` parameter, but the retrieval results also contain information on the `alarm_report` parameter, the LLM elaborates on both parameters in the answer, which is undesired.

6.3 Modularized Evaluation on TIG

The TIG module enhances the ability of LLMs to interact with database, particularly in DB instance-related QA. The benefit of TIG is already highlighted in Table 4. Specifically, we examine the correlation between WinRate and tool invocation success rate. Our findings show that nearly all (93%) responses, where the tool invocations are successful, achieve a win in the final answer comparison. In this section, we further explore the capabilities of existing LLMs and discuss how LLMs can support instance-related QA.

We propose TSA (Tool Selection Accuracy) and TFA (Tool Format Accuracy) to measure tool invocation capabilities. TSA measures whether the correct tools are chosen to solve problems and TFA measures the accuracy of the tool invocation format, i.e., whether the LLM's response aligns with the tool's input (definition in Appendix B).

As shown in Figure 8 (1) The ability of LLMs to select appropriate tools and format input correctly is closely correlated with their overall performance. (2) There is a significant difference in the ability of the tested LLMs to invoke database tools. Comparing Figure 8 and Table 4, we observe that the difference in tool invocation ability is greater than in answering general questions. For instance, the WinRate of LLaMa2 and Yuan in TIG is close to zero, while the performance of Baichuan2 variants approaches one. (3) The capability of LLMs to invoke DB tools is impacted by whether the models underwent instruction-following fine-tuning and alignment, i.e., adjusting the LLMs to align their behaviors and respond in a particular format. E.g. the technical reports for LLaMA2 [42] and Yuan [46] indicate that they lack targeted instruction-following fine-tuning and alignment, producing the worst TSA and TFA results. In contrast, *DQATestbed*implements fine-tuning on DB-related examples to enhance Baichuan-13B's ability to follow DB-specific instructions and obtains the best TSA and TFA results.

7 Conclusion

In this paper, we propose the first comprehensive DBQA benchmark ***DQABench***, which includes an extensive dataset that simulates real-world DB scenarios and a complete testbed that implements the entire DBQA workflow. Our comprehensive evaluation demonstrates the impact of modules such as QCR, RAG, and TIG on DBQA performance. The proposed benchmark dataset, testbed, and findings will guide the future development of LLM-based database applications.

8 Acknowledgments

Chen Lin is the corresponding author. Chen Lin is supported by the National Key R&D Program of China (No. 2022ZD0160501), the Natural Science Foundation of China (No.62372390,62432011).

References

[1] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403* (2023).

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511* (2023).

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[4] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. *arXiv preprint arXiv:2308.14508* (2023).

[5] Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A Lastras. 2024. Api-blend: A comprehensive corpora for training and benchmarking api llms. *arXiv preprint arXiv:2402.15491* (2024).

[6] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. 2023. Accurate medium-range global weather forecasting with 3D neural networks. *Nature* 619, 7970 (2023), 533–538.

[7] Harrison Chase. 2022. LangChain. (2022). https://github.com/hwchase17/langchain

[8] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875* (2022).

[9] Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre FT Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, et al. 2024. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883* (2024).

[10] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3302–3309.

[11] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[13] Kehan Guo, Bozhao Nan, Yujun Zhou, Taicheng Guo, Zhichun Guo, Mihir Surve, Zhenwen Liang, Nitesh Chawla, Olaf Wiest, and Xiangliang Zhang. 2025. Can llms solve molecule puzzles? a multimodal benchmark for molecular structure elucidation. *Advances in Neural Information Processing Systems* 37 (2025), 134721–134746.

[14] Jakob Hauser, Dániel Kondor, Jenny Reddish, Majid Benam, Enrico Cioni, Federica Villa, James Bennett, Daniel Hoyer, Pieter Francois, Peter Turchin, et al. 2025. Large Language Models' Expert-level Global History Knowledge Benchmark (HiST-LLM). *Advances in Neural Information Processing Systems* 37 (2025), 32336–32369.

[15] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 507–517.

[16] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300* (2020).

[17] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060* (2020).

[18] Jiaming Ji, Mickel Liu, Juntao Dai, Xuehai Pan, Chi Zhang, Ce Bian, Chi Zhang, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. BeaverTails: Towards Improved Safety Alignment of LLM via a Human-Preference Dataset. *arXiv preprint arXiv:2307.04657* (2023).

[19] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[20] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983* (2023).

[21] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* (2017).

[22] Jean Lee, Nicholas Stevens, Soyeon Caren Han, and Minseok Song. 2024. A Survey of Large Language Models in Finance (FinLLMs). *arXiv preprint arXiv:2402.02315* (2024).

[23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[24] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[25] Shuyue Stella Li, Vidhisha Balachandran, Shangbin Feng, Jonathan S Ilgen, Emma Pierson, Pang Wei Koh, and Yulia Tsvetkov. 2024. MediQ: Question-Asking LLMs and a Benchmark for Reliable Interactive Clinical Reasoning. *arXiv preprint arXiv:2406.00922* (2024).

[26] Jie Liu and Barzan Mozafari. 2024. Query Rewriting via Large Language Models. *arXiv preprint arXiv:2403.09060* (2024).

[27] Yang Liu, Jiahuan Cao, Chongyu Liu, Kai Ding, and Lianwen Jin. 2024. Datasets for large language models: A comprehensive survey. *arXiv preprint arXiv:2402.18041* (2024).

[28] Yi Luo, Zhenghao Lin, Yuhao Zhang, Jiashuo Sun, Chen Lin, Chengjin Xu, Xiangdong Su, Yelong Shen, Jian Guo, and Yeyun Gong. 2024. Ensuring Safe and High-Quality Outputs: A Guideline Library Approach for Language Models. *CoRR* abs/2403.11838 (2024). arXiv:2403.11838

[29] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283* (2023).

[30] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511* (2022).

[31] OpenAI. 2022. OpenAI: Introducing ChatGPT. (2022). https://openai.com/blog/chatgpt

[32] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). doi:10.48550/arXiv.2303.08774 arXiv:2303.08774

[33] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2024).

[34] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350* (2022).

[35] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. *arXiv preprint arXiv:1806.03822* (2018).

[36] Reddit. 2015. Reddit Comments Dataset. https://www.reddit.com/r/datasets/.

[37] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).

[38] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294* (2023).

[39] Hao Sun, Zhexin Zhang, Jiawen Deng, Jiale Cheng, and Minlie Huang. 2023. Safety Assessment of Chinese Large Language Models. *arXiv preprint arXiv:2304.10436* (2023).

[40] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *arXiv preprint arXiv:2307.07697* (2023).

[41] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[42] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[43] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[44] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 1–26.

[45] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[46] Shaohua Wu, Xudong Zhao, Shenling Wang, Jiangang Luo, Lingjun Li, Xi Chen, Bing Zhao, Wei Wang, Tong Yu, Rongguo Zhang, et al. 2023. YUAN 2.0: A Large Language Model with Localized Filtering-based Attention. *arXiv preprint*

arXiv:2311.15786 (2023).
[47] Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. 2023. Smartplay: A benchmark for llms as intelligent agents. *arXiv preprint arXiv:2310.01557* (2023).
[48] Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. 2024. FOFO: A Benchmark to Evaluate LLMs' Format-Following Capability. *arXiv preprint arXiv:2402.18667* (2024).
[49] Qianqian Xie, Weiguang Han, Zhengyu Chen, Ruoyu Xiang, Xiao Zhang, Yueru He, Mengxi Xiao, Dong Li, Yongfu Dai, Duanyu Feng, et al. 2025. Finben: A holistic financial benchmark for large language models. *Advances in Neural Information Processing Systems* 37 (2025), 95716–95743.
[50] Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, et al. 2023. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449* (2023).
[51] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305* (2023).
[52] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 5754–5764. https://proceedings.neurips.cc/paper/2019/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html
[53] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=WE_vluYUL-X
[54] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).
[55] Mingze Yuan, Peng Bao, Jiajia Yuan, Yunhao Shen, Zifan Chen, Yi Xie, Jie Zhao, Yang Chen, Li Zhang, Lin Shen, et al. 2023. Large Language Models Illuminate a Progressive Pathway to Artificial Healthcare Assistant: A Review. *arXiv preprint arXiv:2311.01918* (2023).
[56] Rui Yuan, Wanting Hao, and Chun Yuan. 2024. Benchmarking AI in Mental Health: A Critical Examination of LLMs Across Key Performance and Ethical Metrics. In *International Conference on Pattern Recognition*. Springer, 351–366.
[57] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).
[58] Hongbo Zhang, Junying Chen, Feng Jiang, Fei Yu, Zhihong Chen, Jianquan Li, Guiming Chen, Xiangbo Wu, Zhiyi Zhang, Qingying Xiao, et al. 2023. Huatuogpt, towards taming language model to be a doctor. *arXiv preprint arXiv:2305.15075* (2023).
[59] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, et al. 2024. RAGLAB: A Modular and Research-Oriented Unified Framework for Retrieval-Augmented Generation. *arXiv preprint arXiv:2408.11381* (2024).
[60] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. 2023. Llm as dba. *arXiv preprint arXiv:2308.05481* (2023).
[61] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: Database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454* (2023).
[62] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Science and Engineering* (2024), 1–10.

A Testbed Impletation Details

A.1 Pre-training and Fine-tuning

(1) Pre-training. We extensively collect pre-training corpora related to databases, comprising approximately 47,000 entries each in Chinese and English, totaling around 100 million tokens. This corpus includes major database textbooks, official documentation of various database products, and selected authoritative reports and articles on databases.

(2) Fine-tuning. We propose a sequential fine-tuning strategy, including three stages. We prioritize the fine-tuning sequence based on the crucial abilities in DB problem-solving. For instance, the first fine-tuning stage focuses on enhancing the LLM's NL2SQL and table understanding ability using NL2SQL data like Spider [54] because it is fundamental in DB tasks. In the second fine-tuning stage, a mixture of different fine-tuning data is adopted. The fine-tuning data includes (1) general conversational datasets like Stanford Alpaca [41] to mitigate the LLM's forgetting of general dialogue skills, and (2) reformulated questions from ***DQABench*** using corresponding prompts in the PTE module to enhance the LLM's understanding of the prompt template. The last fine-tuning stage focuses on enhancing the alignment of LLM's final response with DB experts in terms of quality and format, by using answer cases written by DB experts in Section 3. The specific settings will be detailed in Section 5.

A.2 Question Classification Routing

In this paper, we implement and evaluate three methods of QCR modules to explore the better design paradigm.

(1) LLM-based Classification Method. We use a prompt, which is designed to elicit a classification response from GPT-4. [1]

(2) Classifier. We train an XLNet-based [52] classifier. We construct the training data[2] where each question is labeled as "unsafe", "safe but irrelevant", "DB general", "product-specific", or "instance-specific". The positive samples for the "unsafe" category are collected from Safety-Prompts [39] and BeaverTails-Evaluation [18]. The "safe but irrelevant" samples are collected from Alpaca [41] and Longbench [4]. The rest three categories are from the training set of ***DQABench*** (which does not overlap with the test set).

(3) Hierarchical Classifier. Training a single function to predict all possible labels is more difficult. Furthermore, a "flat" classifier method requires a balanced amount of training queries for each class. Alternatively, we train a hierarchical classifier, which first classifies safe and unsafe questions and then classifies the safe questions into four sub-classes. We use an independent XLNet-based [52] classifier at each level.

A.3 Retrieval Augment Generation

The RAG implementation utilizes the multilingual text encoding model bge-m3 with specific experimental parameters. The system is configured with a temperature of 0.0 and a top_p value of 1.0. For document retrieval, the self_rag method retrieves 5 documents and includes an additional filtering step to ensure the usefulness of the content, while the other methods retrieve 3 documents each. All methods enforce a maximum answer length of 500 tokens, with the active_rag method further restricting each sentence to 50 tokens. Additionally, active_rag is set with a filter probability of 0.8 and a masked probability of 0.4. Both the iterative_rag and self_ask methods are limited to a maximum of 3 iterations. In the self_rag method, the weight parameters are configured as follows: w_rel at 1.0, w_sup at 1.0, and w_use at 0.5.

A.4 Tool Invocation Generation

We implement a Chain of Thought (COT) prompt template following ReAct [53]. This prompt template encourages the LLM to think in a loop according to the following chain of thought the same as the instance-specific prompt on **??**. The LLM first outputs a COT

[1] All prompts used on this paper can be found on [link]
[2] The sources and statistics of the dataset for these classifiers are detailed in [link].

Table 8: Supported types of common DB tools

Objective	Type	Functionality
Data Modeling	Schema	Obtain database table structure, constraints, etc.
	Selection	Return SQL execution results of retrieving specific data from the database, computing data distribution, etc.
Database Monitoring	Resource	Obtain information about CPU usage, memory, disk IO, etc.
	Workload	Workload analysis, slow question identification, etc.
	Status	Detailed information about the current indexes, views, knob settings, etc.
Optimizing	Tuning	Identify optimization opportunities by advising indexes, setting knobs, etc.

with the tool it wants to invoke and its input. The tool trigger interrupts the LLM's output, while the TIG module identifies the tool name (following "Action:") from the pool in Table 8. If found, it calls the tool using the input specified after "Action_Input:". Upon interacting with the database, the tool outputs results, formatted as text and appended to the LLM's output afte "Observation:". We optimize these outputs by (1) filtering relevant content and (2) converting structured data into Markdown. After tool execution, the TIG module resumes the LLM's process, which uses the "Observation" results to decide whether to invoke additional tools. The cycle continues until the LLM has enough information to output a final answer.

B Experiments Setting

Evaluation pipeline. Directly using the LLMs can hardly generate satisfying answers for DB questions in ***DQABench***, even with prompt engineering. The workflow in Section 4, i.e., with auxiliary techniques such as QCR, RAG, and TIG, can improve the answer quality of each tested LLM. Thus, this section presents the results generated by LLMs through the entire workflow implementing all modules of *DQATestbed*. LLM performances without the auxiliary techniques will be further investigated in Section 6. Since each module can adopt various strategies that may introduce bias when comparing the performance between the LLMs, we use a standard pipeline with fixed intermediate output to obtain the best output of different LLMs. The details of the standard pipeline are as follows.

(1) Question Routing with Ground-truth Class Label. Question classification accuracy impacts answer generation performance. Correctly associating "Why is SQL query execution slow?" with the 'instance-specific' label is critical for triggering DB tools and generating targeted responses. We analyze different classification strategies in Section 6. Here, we use ground-truth labels to construct optimal prompts for the core LLM.

(2) Generation with Ground-truth Retrieved Knowledge. Product-specific questions require LLMs to access retrieval documents containing product information. As shown in Section 6, retrieval precision directly affects answer quality. We append ground-truth fine-grained retrieval text (correct text blocks) to prompts for evaluation.

(3) Generation with Ground-truth Tool Output. Instance-specific questions evaluate LLMs' ability to plan and use DB tools. However, tool outputs often deviate from ground truth, causing hallucinated tools/APIs and evaluation challenges. For these queries, we implement a "Thought-Action-Action_Input-Observation" process.

Testing questions. The testing questions consist of *(1) DB general questions* are fundamental concepts in the database domain, *(2) product-specific questions* are about the database product 'openGauss', where the external retrieval documents are openGauss latest documentation as of April 2024. The advantage of 'openGauss' is that our evaluated LLMs have not shown any signs of being specifically trained on the latest detailed documentation of this product, effectively avoiding unfair evaluation due to potential data leakage. *(3) instance-specific questions* are created on the widely recognized database benchmarks TPC-H, TPC-C, and TPC-DS.

Metrics We adopt two evaluation metrics, WinRate and MCA (Multiple Choice Accuracy), to measure the quality of end-to-end answer generation.

(1) WinRate. Defined as shown in 5. We calculate WinRate as follows:

$$WinRate = \frac{N_{r=1}}{N_{r=1} + N_{r=-1}}, \quad \text{where} \begin{cases} r = 1 & \text{if } M \text{ wins,} \\ r = 0 & \text{if } M \text{ ties,} \\ r = -1 & \text{if } M \text{ loses,} \end{cases} \tag{1}$$

where N_r represents the number of comparisons where the judge GPT-4 considers case r, and M is the model to be evaluated.

(2) MCA (Multiple-Choice Accuracy). To complement the objective LLM-based evaluation WinRate, we also provide subjective evaluations on DB-general questions. This metric measures the accuracy of all multiple-choice questions following $MCA = \sum_i m_{i,i} / \sum_i \sum_j m_{i,j}$), where $m_{i,j}$ is the number of answers that the ground truth choice is i and the LLM's output is j, $i, j \in \{A, B, C, D, others\}$. For MCA, we prompt the LLM to output one letter; any deviation is classified as $others$ and considered a categorization error.

(3) TSA measures whether the correct tools are chosen to solve problems. It is essential to consider the order of actions to measure the DB-specific planning ability. For example, the input of the succeeding tool is usually based on and formulated from the preceding tool's output. Therefore, if there is an error in the current tool invocation, the subsequent invoked tools will no longer be included in the metric calculation. Specifically, the TSA (Tool Selection Accuracy) is defined as: $TSA = \sum_{1 \le i \le \min k_j, I\{t_{k_j,j}\}=0, j} I\{t_{i,j}, j\} \big/ \sum_j k_j$, where $t_{i,j}$ is $i-$th tool for the query j, $I\{\}$ is an indicator function that returns whether the tool is labeled in the tool annotation in Section 3, and k_j means the number of LLM tool invocations.

(4) TFA measures the accuracy of the tool invocation format, i.e., whether the LLM's response aligns with the tool's input. Due to the diversity and subjectivity of tool format requirements, particularly in the generalized tool QA, it is challenging to assess format compliance using predefined rules. Therefore, we employ GPT-4 as an expert adjudicator model to judge whether tool invocations meet the format requirements. Similarly, we consider the order of tools. Specifically, the TFA (Tool Format Accuracy) is defined as: $TFA = \sum_{1 \le i \le \min k_j, G\{t_{k_j,j}\}=0, j} G\{t_{i,j}\} \big/ \sum_j k_j$, where $t_{i,j}$ is the $i-$th tool for the query j, $G\{\}$ is the output of GPT-4 that decides whether the tool input (the content after "Action_Input") is correct, and k_j means the number of LLM tool invocations.