LLMs4SQL Group Project Report

1. Team Information

- **Siyu Xie (72542207)** Undergraduate Major: Food Science and Engineering
- **Xinfang Zhang (72542152)** Undergraduate Major: Information Management and Information Systems
- **Jingyi Dong (72542072)** Undergraduate Major: Big Data Management and Application
- **Wenyue Yang (72542268)** Undergraduate Major: Information Management and Information Systems
- **Jingwen Luo (72542176)** Undergraduate Major: Information and Computing Science

2. Contribution Statement

- **Siyu Xie (72542207)**:
  - Large model cloud platform interface code encapsulation
  - `missing_token` evaluation result analysis + report
- **Xinfang Zhang (72542152)**:
  - Model inference code pipeline encapsulation
  - `syntax_error` evaluation result analysis + report
- **Jingyi Dong (72542072)**:
  - Paper search + paper analysis
  - `query_performance` evaluation result analysis + report
- **Wenyue Yang (72542268)**:
  - Result evaluation code pipeline encapsulation
  - `query_equality` evaluation result analysis + report
- **Jingwen Luo (72542176)**:
  - Data preprocessing code + data mapping code
  - Report integration and writing + code repository integration

> Note: All team members conducted **original paper reading and analysis + optimization and innovation solution** design thinking, and implemented **code for different modules** during the project process.

3. Project Overview

3.1 Background & Motivation

Large Language Models (LLMs) have demonstrated strong performance in natural language processing, code generation, and other domains in recent years. However, questions remain about whether they can "truly understand" Structured Query Language (SQL). Structured query language imposes strict requirements on syntax, semantics, context, and execution logic. The original paper proposes a comprehensive evaluation of LLMs' "understanding capabilities" through a series of SQL-centric tasks.

3.2 Core Paper

- **Paper Title**: *Evaluating SQL Understanding in Large Language Models* (https://arxiv.org/abs/2410.10680)
- **Authors**: Ananya Rahaman, Anny Zheng, Mostafa Milani, Fei Chiang, Rachel Pottinger (https://arxiv.org/abs/2410.10680)
- **Publication/Submission Date**: October 2024 (arXiv); subsequently accepted by EDBT 2025 conference (Volume 28, pp. 909-921) (https://chatgpt.com/c/6936999c-6c74-8324-b08e-59372426e2c5)
- **Research Objectives**:
  To evaluate LLM performance on SQL tasks, examining their strengths and weaknesses across four capability dimensions: recognition, context, semantics, and coherence. The paper designed a series of tasks including syntax error detection, missing token identification, query performance prediction, query equivalence checking, and SQL → natural language explanation.
- **Main Findings**:
  While certain models (such as GPT-4) perform well on basic tasks (recognition / context), all models exhibit significant deficiencies in deeper semantic understanding and coherence (especially equivalence judgment and performance estimation). That is, current LLMs still have obvious limitations in their ability to "truly understand complex SQL semantics and logic."

3.3 Motivation & Objectives for Reproduction

- **Reproduction Motivation**:
  i. Although the original paper provides task definitions and evaluation approaches, it lacks a complete reproducible pipeline—prompt design, output parsing, evaluation workflow, etc. are not publicly available.
  ii. In practical deployment and scientific research reproduction, a structured, engineered, and extensible framework is needed for unified evaluation across different models / different prompts / different tasks.
  iii. Through systematic reconstruction and expansion (finer-grained task dimensions, structured outputs, stable sampling, unified LLM server interfaces), construct a "trustworthy SQL understanding capability benchmark"—which has practical value for future research and system applications.
- **Reproduction Objectives**:
  i. Based on the paper's task definitions, reconstruct and expand tasks including syntax error detection, missing token identification, query performance prediction, and query equivalence analysis.
  ii. Design structured prompts + JSON schema outputs + stable sampling + unified LLM server interfaces.
  iii. Construct inference pipeline + evaluation pipeline to enable automatic calculation of evaluation results (binary classification, multi-class classification, location prediction, F1 / MAE / Hit-Rate, etc.).
  iv. Support plug-and-play testing of multiple LLMs.
  v. Since the reproducibility of the original paper's results is questionable, this project only reproduces the pipeline independently for comparative analysis based on the original paper's task definitions and data, without relying on the original paper's experimental results.

4. Technical Design

4.1 Overview of Paper Methodology

The original paper proposes five categories of SQL-based tasks (syntax error detection, missing token identification, query performance prediction, query equivalence checking, query explanation), with the following core technical approach:

1. **Task Definition**: Each task clearly defines inputs (SQL queries) and outputs (error labels, performance categories, equivalence labels, etc.).
2. **Evaluation Metrics**: Primarily uses statistical metrics such as Precision, Recall, F1-Score, with some tasks introducing MAE, Hit Rate, etc. to measure model prediction accuracy.
3. **Prompt Usage**: The paper guides model answer generation through simple natural language prompts, but does not specify output structure or JSON schema, nor unified randomness sampling strategies.
4. **Data and Experiments**: Data sources include SDSS, SQLShare, Join-Order, and Spider; experimental results partially involve manual evaluation, with insufficient reproducibility and automation.

4.2 Implementation Strategy of This Project

To address the limitations of the original paper's methodology, this project designs a complete, reproducible technical implementation framework, including:

1. **Data Processing and Standardization**
   - Collect and clean the original paper's datasets, unifying fields and table structure information.
   - Perform tokenization and structured annotation on queries to facilitate subsequent task parsing and localization.
2. **Prompt Engineering**
   - Design zero-shot and few-shot prompts to ensure model outputs conform to JSON schema for automatic parsing.
   - Introduce role, workflow constraints to guide models in generating coherent, parseable responses.
3. **Inference Pipeline**
   - Implement SQL-centric task inference pipeline supporting unified evaluation across different models / different prompts / different tasks.
   - Construct unified LLM interface (Doubao / Qwen / Deepseek / GLM, etc.), supporting thinking / non-thinking modes.
   - Support batch inference, randomness control, and reproducible sampling.
4. **Evaluation Pipeline**
   - Perform automatic evaluation of different task types including binary classification, multi-class classification, location prediction, etc.
   - Automatically calculate Precision / Recall / F1 / MAE / Hit Rate metrics, generating visual result tables and analysis reports.
5. **Result Analysis**
   - Provide model performance analysis at the task dimension, revealing model limitations in complex semantic and equivalence tasks.
   - Do not compare with original paper experimental results to ensure data reliability and experimental reproducibility.

4.3 Deviation Statement from Paper Methodology

- This project does not directly use the paper's manual evaluation results, but recalculates metrics through a unified, automated pipeline.
- Since the original paper does not provide specific model versions, prompts, or sampling details, this project's experimental environment differs slightly from the paper, therefore no direct numerical comparisons are made.
- The following models were selected for experimental comparative analysis:
  - **Doubao-Seed-1.6-251015** ~ Source: ByteDance's latest controllable thinking model
  - **qwen3-next-80b-a3b-instruct** ~ Source: Alibaba / Tongyi Qwen's latest non-thinking model in open-source series, MOE architecture
  - **GLM-4.6** ~ Source: Zhipu AI's latest controllable thinking model
  - **DeepSeek-V3.1-Terminus** ~ Source: Controllable thinking model launched by DeepSeek

    - **DeepSeek-V3.1-Terminus (Thinking mode enabled)** ~ Source: Controllable thinking model launched by DeepSeek
- Output formats, evaluation workflows, and task divisions were optimized in this project to improve reproducibility and engineering level.

5. Algorithmic / System Implementation

5.1 Core Algorithm Description

This project constructs a **unified LLM SQL understanding evaluation system** with the following core design:

1. **Multi-Platform LLM Interface Encapsulation**
    - `LLMServer` provides a unified interface supporting multiple platforms including Doubao, Qwen, SiliconFlow, and standard OpenAI interfaces.
    - Supports `chat`, `vision chat`, `embedding` modes, with configurable model reasoning capability.
    - Backend uniformly encapsulates API requests to ensure reproducible inference workflow.
2. **Inference Pipeline (`Inference` class)**
    - Input: SQL query + task type (`InferType`)
    - Output: Structured results conforming to JSON schema for automated parsing.
    - Supports batch processing and multi-threaded inference (`max_workers` controls parallel count).
    - Configurable inference strategy: whether to enable reasoning / thinking mode.
3. **Evaluation Pipeline (`EvaluateTool` class)**
    - Automatically evaluates different task types including binary classification, multi-class classification, location prediction, etc.
    - Metrics support: Precision / Recall / F1 / MAE / Hit Rate.
    - Supports Macro F1 calculation for global performance measurement in multi-class tasks.
    - Automatically loads evaluation data, outputs visual results and analysis reports.
4. **Configuration-Driven Design**
    - Uses YAML configuration files for unified management of model parameters, inference strategies, evaluation tasks, and data paths.
    - Supports flexible switching of models and tasks without modifying core code.

Example configuration (`infer.yaml`):

```yaml
model:
  model_type: doubao
  base_url: https://ark.cn-beijing.volces.com/api/v3/
  api_key: api_key
  reasoning_ability: True

inference:
  model_name: model_name
  model_identifier: NULL
  reasoning: False
  max_workers: 10
```

`eval.yaml`:

```yaml
evaluation:
  infer_type: syntax_error
  data_dir: outputs/syntax_error
  model_list: ['DeepSeek-V3.1-Terminus-Thinking', 'GLM-4.6', 'DeepSeek-V3.1-Terminus', 'Doubao-Seed-1.6-2510
```

5.2 Key Data Structures

- **`Inference` class**
  - `llms: LLMServer` — LLM interface object
  - `infer_type: InferType` — inference task type
  - `model_name / model_identifier` — model identifier
  - `max_workers` — parallel inference thread count
- **`EvaluateTool` class**
  - `dataset` — dataset corresponding to the task
  - `infer_type` — evaluation task type
  - `metrics` — automatically calculates Precision / Recall / F1 / MAE / Hit Rate
- **Inference Output Example**:

```json
{
  "syntax_error": "YES/NO",
  "syntax_error_type": <type>
}
```

5.3 Correctness Verification Methods

- **Data Consistency Check**: Perform token / schema alignment between input SQL queries and annotated data to ensure valid inference input.
- **Result Parsing Validation**: Perform strict schema validation on JSON outputs to avoid metric bias caused by parsing errors.
- **Metric Comparison**: For multi-class tasks, use Macro F1 for global measurement to ensure direct comparability across different models.
- **Multi-Model & Multi-Task Retesting**: Use multi-threaded parallel validation to ensure inference speed and stability and reproducibility of evaluation results.

6. Evaluation and Results

6.1 Experimental Setup

- **Hardware Environment**: Apple Mac M3 Pro, 18-core CPU, 36GB memory (integrated / parallel acceleration)
- **Software Environment**:
  - Python 3.10
  - Data processing libraries including `pandas / numpy / yaml / tqdm /matplotlib`
  - `openai` library for model API calls.

- Custom `LLMServer`, `Inference`, `EvaluateTool` modules

6.2 Datasets Used

Dataset	Source	Query Count	Description
SDSS	Sloan Digital Sky Survey (2023)	285	Astronomical query data containing various table joins, filter conditions, and aggregation operations. Used for syntax error detection, missing token identification, query performance prediction, and query equivalence checking.
SQLShare	SQLShare Platform	251	SQL queries for educational and research purposes covering different complexities, testing LLM understanding capabilities of multi-table joins, subqueries, and aggregation functions.
Join-Order	Join-Order Benchmark	157	Query set focused on SQL join order and execution performance, primarily used for syntax detection, missing token and equivalence checking.

> All data underwent preprocessing, including standardizing table/column names, tokenization, task annotation, and JSON schema conversion to ensure automatic evaluation.

6.3 Evaluation Performance Metrics

Different metrics were adopted for automation evaluation based on task types:

Task	Performance Metrics	Description
Syntax Error Detection	Precision / Recall / F1-Score	Binary / multi-class performance measurement
Missing Token Identification	Precision / Recall / F1-Score / MAE / Hit Rate	Simultaneously measures location prediction accuracy of missing positions
Query Performance Prediction	Precision / Recall / F1-Score	Classification accuracy for high-cost / low-cost queries
Query Equivalence Checking	Precision / Recall / F1-Score / Macro F1	Multi-class task supporting equivalence category mapping
Query Explanation	Qualitative Analysis	Converting SQL to natural language descriptions, evaluating readability and semantic accuracy, not used as core metric

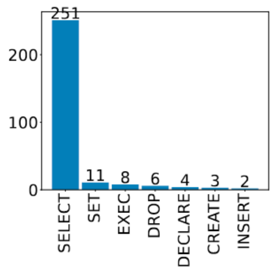
> All tasks automatically calculate metrics through unified pipeline to ensure reproducibility and multi-model comparability.

6.4 Experimental Result Analysis

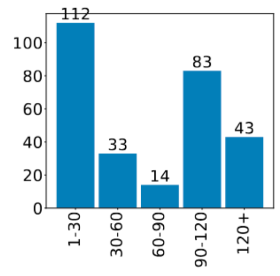
> No comparison with original paper experimental results to ensure data reliability and experimental reproducibility.

6.4.1 Data Preprocessing and Visualization

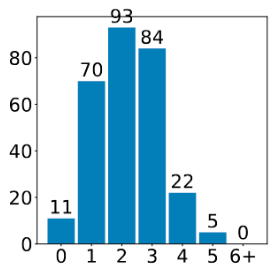
- **SDSS Statistics:**



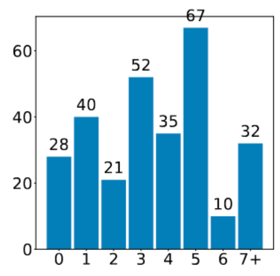
(a) query_type



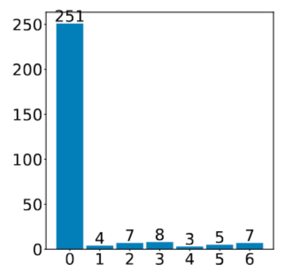
(b) word_count



(c) table_count

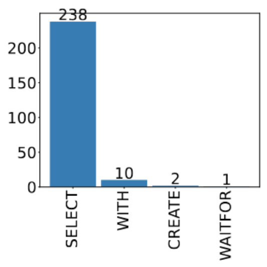


(d) predicate_count

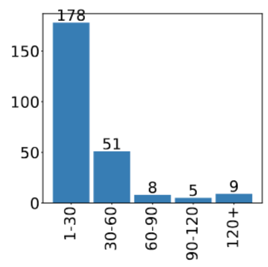


(e) nestedness

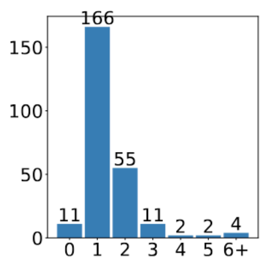
- **SQLSHARE Statistics:**



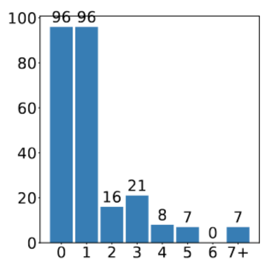
(a) query_type



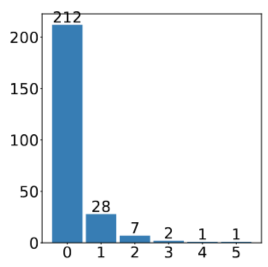
(b) word_count



(c) table_count

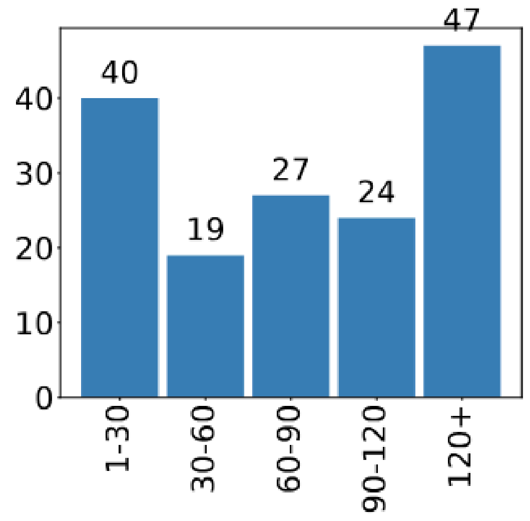


(d) predicate_count

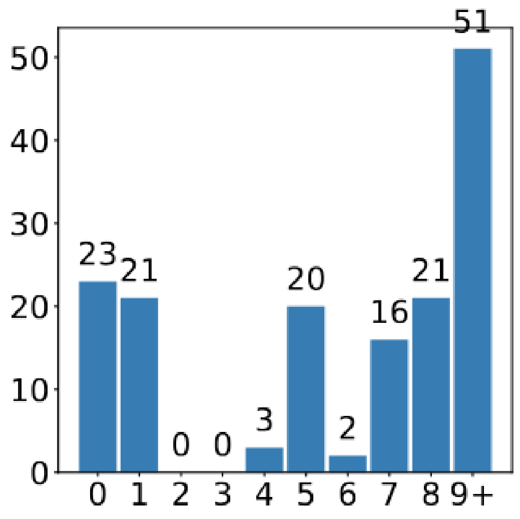


(e) nestedness

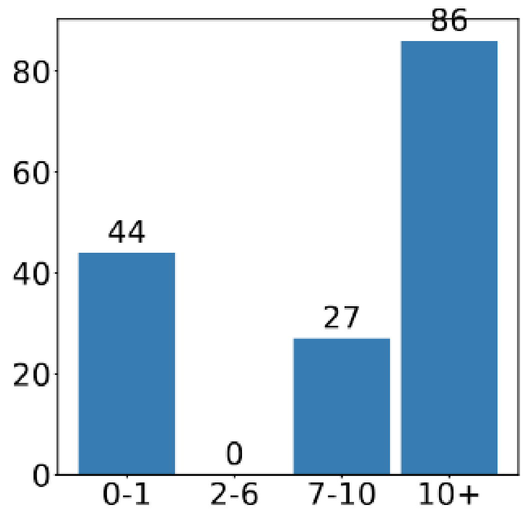
- **Join-Order Statistics:**



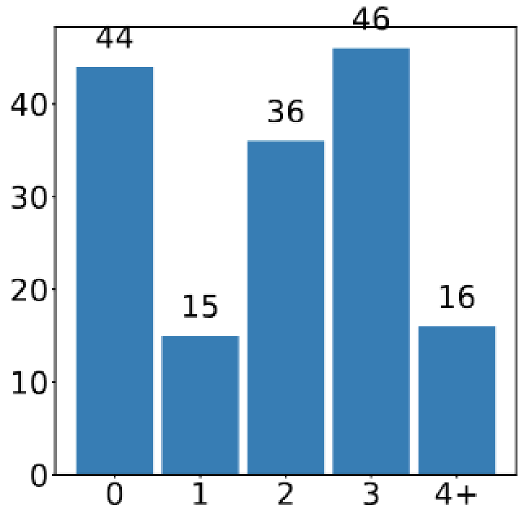
(a) word_count



(b) table_count



(c) predicate_count

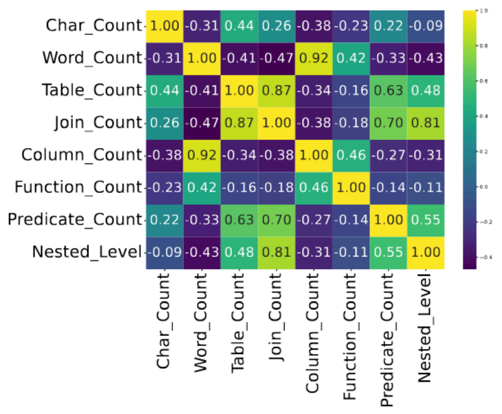


(d) function_count

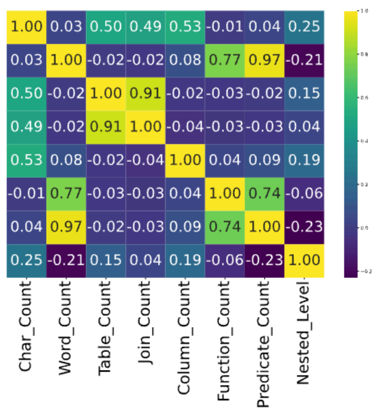
The above images show the statistical attributes of data volumes. Each figure is a histogram displaying query count on the y-axis and query attributes on the x-axis, where `x` represents the value range of attributes. For example, Figure 1b shows the number of queries (y-axis) across different query length (word_count) ranges. These charts indicate that SDSS and SQLShare contain more complex queries involving multiple tables and broader predicate types. In contrast, Join-Order queries are simpler and less nested. In terms of query length (word_count), SDSS and Join-Order queries are longer than SQLShare.

Since strong correlations may exist between pairwise attributes leading to redundancy and inefficiency, we used Pearson correlation coefficient to examine correlations between pairwise query attributes, adopting a threshold of 0.7 to indicate strong correlation:

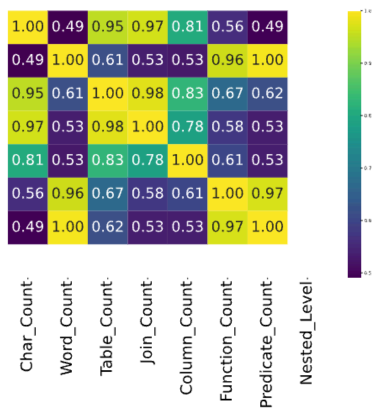
- **The pairwise correlations between query attributes under each dataset:**



(a) SDSS



(b) SQLShare

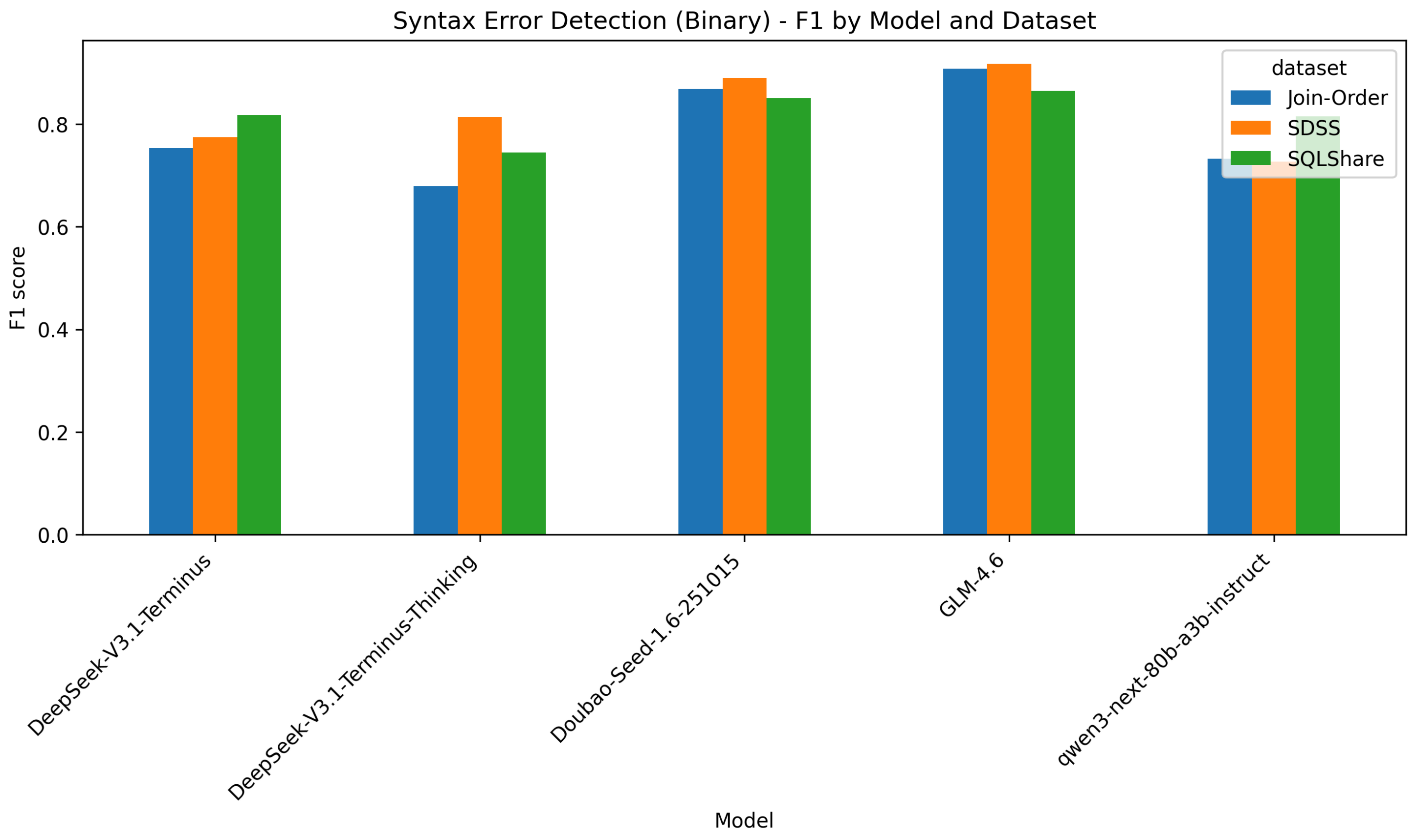


(c) Join-Order

6.4.2 Syntax Error Detection Performance Comparison

- i. `syntax_error` metrics

Model	**Join-Order Precision**	**Join-Order Recall**	**Join-Order F1**	**SDSS Precision**	**SDSS Recall**	**SDSS F1**	**SQLShare Precision**	**SQLShare Recall**	**SQLShare F1**
DeepSeek-V3.1-Terminus-Thinking	**1.00**	0.51	0.68	0.99	0.69	0.81	0.93	0.62	0.74
GLM-4.6	0.92	**0.90**	**0.91**	0.99	0.86	**0.92**	**0.96**	0.79	**0.86**
DeepSeek-V3.1-Terminus	0.97	0.61	0.75	0.95	0.65	0.77	0.92	0.74	0.82
Doubao-Seed-1.6-251015	0.86	0.88	0.87	0.90	**0.88**	0.89	0.91	**0.80**	0.85
qwen3-next-80b-a3b-instruct	**1.00**	0.58	0.73	0.96	0.58	0.73	**0.96**	0.71	0.81

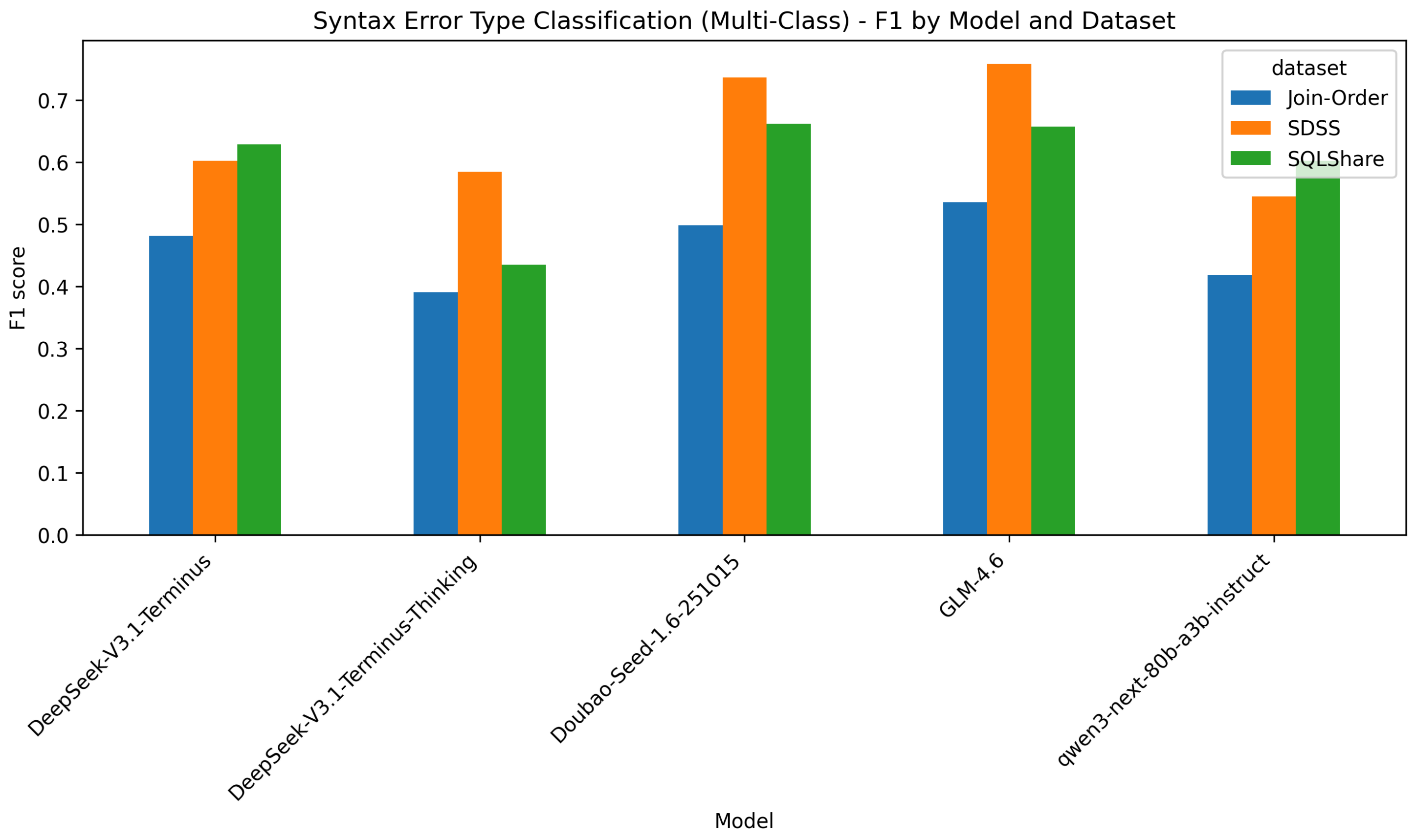


- **Binary Classification Analysis of Syntax Error Detection**:
  - GLM-4.6 and Doubao-Seed-1.6-251015 achieved the highest and most stable F1 values across all datasets. DeepSeek-V3.1-Terminus-Thinking performed notably poorly on Join-Order.
  - All datasets showed relatively high F1 scores, with problems on the SDSS dataset being slightly easier for models.
  - Overall performance ranking: GLM-4.6 > Doubao-Seed-1.6-251015 > DeepSeek-V3.1-Terminus > qwen3-next-80b-a3b-instruct > DeepSeek-V3.1-Terminus-Thinking

- ii. `syntax_error_type` metrics

Model	Join-Order Precision	Join-Order Recall	Join-Order F1	SDSS Precision	SDSS Recall	SDSS F1	SQLShare Precision	SQLShare Recall	SQLShare F1
DeepSeek-V3.1-Terminus-Thinking	0.52	0.35	0.39	0.78	0.52	0.58	0.70	0.40	0.43
GLM-4.6	0.59	**0.56**	**0.54**	**0.82**	0.71	**0.76**	0.74	0.61	**0.66**
DeepSeek-V3.1-Terminus	0.64	0.43	0.48	0.81	0.53	0.60	0.72	0.57	0.63

Model	Join-Order Precision	Join-Order Recall	Join-Order F1	SDSS Precision	SDSS Recall	SDSS F1	SQLShare Precision	SQLShare Recall	SQLShare F1
Doubao-Seed-1.6-251015	0.60	0.51	0.50	0.76	**0.73**	0.74	0.73	**0.62**	**0.66**
qwen3-next-80b-a3b-instruct	**0.68**	0.38	0.42	0.79	0.46	0.55	**0.77**	0.54	0.60



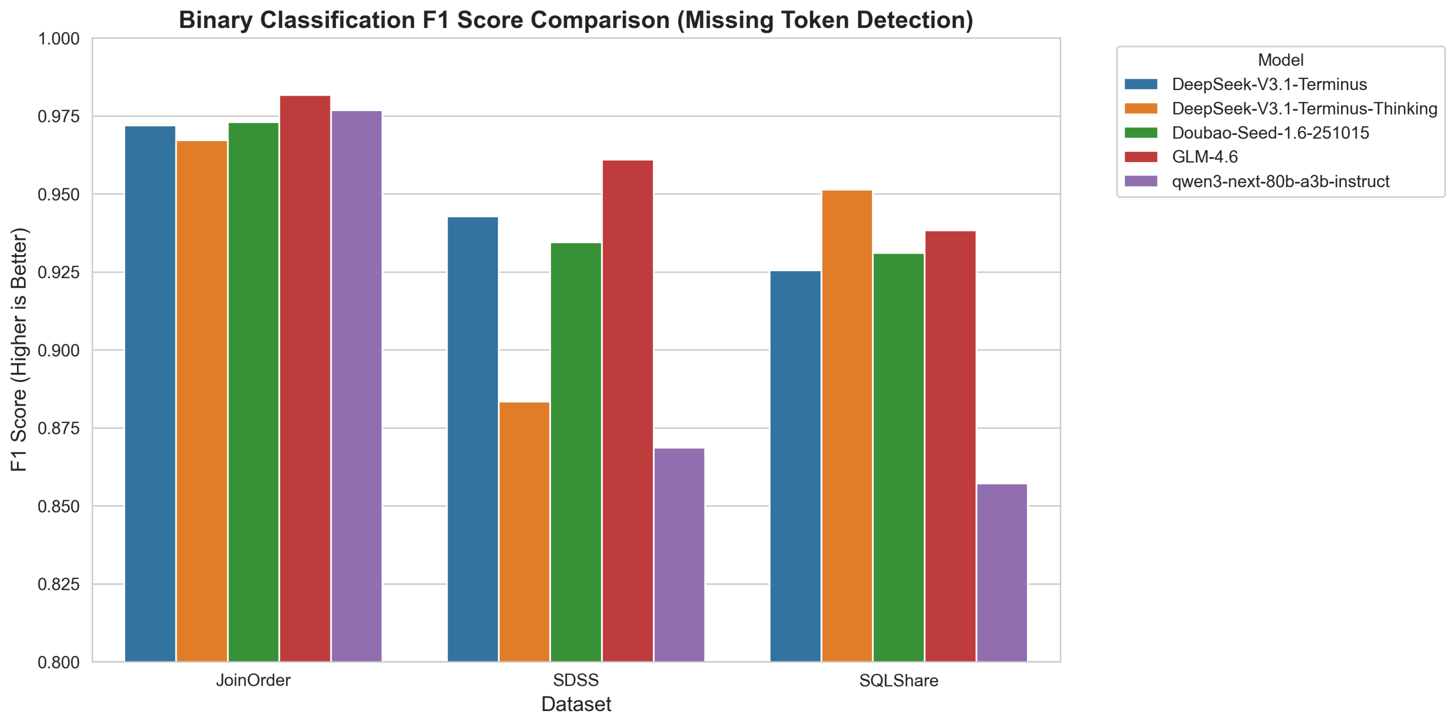
- **Multi-class Analysis of Syntax Error Detection**:
  - GLM-4.6 and Doubao-Seed-1.6-251015 also lead. DeepSeek-V3.1-Terminus-Thinking performed significantly poorly.

- iii. **Conclusions**:
  - **Difficulty between binary and multi-class**: Binary classification is much easier; all models achieve F1 values of 0.75–0.90. Multi-class F1 values drop significantly (within 0.40–0.65 range).
  - **Model capability comparison**: GLM-4.6 is overall the strongest. Doubao-Seed-1.6-251015 demonstrates competitive performance, often approaching GLM-4.6. DeepSeek-Thinking consistently underperforms, suggesting that chain-of-thought reasoning is not helpful for syntax-level tasks.
  - **Dataset difficulty**: Across both tasks, SDSS is the easiest dataset. Join-Order is the most difficult, especially for multi-class tasks.

6.4.3 Missing Token Identification Performance Comparison

- i. `missing_token` metrics

Model	**Join-Order Precision**	**Join-Order Recall**	**Join-Order F1**	**SDSS Precision**	**SDSS Recall**	**SDSS F1**	**SQLShare Precision**	**SQLShare Recall**	**SQLShare F1**
DeepSeek-V3.1-Terminus-Thinking	0.98	0.95	0.97	0.97	0.81	0.88	0.99	0.92	0.95
GLM-4.6	0.97	0.99	0.98	0.97	0.95	0.96	0.89	**0.99**	0.94
DeepSeek-V3.1-Terminus	0.98	0.96	0.97	0.91	0.97	0.94	0.95	0.91	0.93
Doubao-Seed-1.6-251015	0.95	**1.00**	**0.97**	0.90	0.97	0.93	0.88	0.98	0.93
qwen3-next-80b-a3b-instruct	**0.98**	0.97	0.98	0.85	0.89	0.87	0.93	0.80	0.86

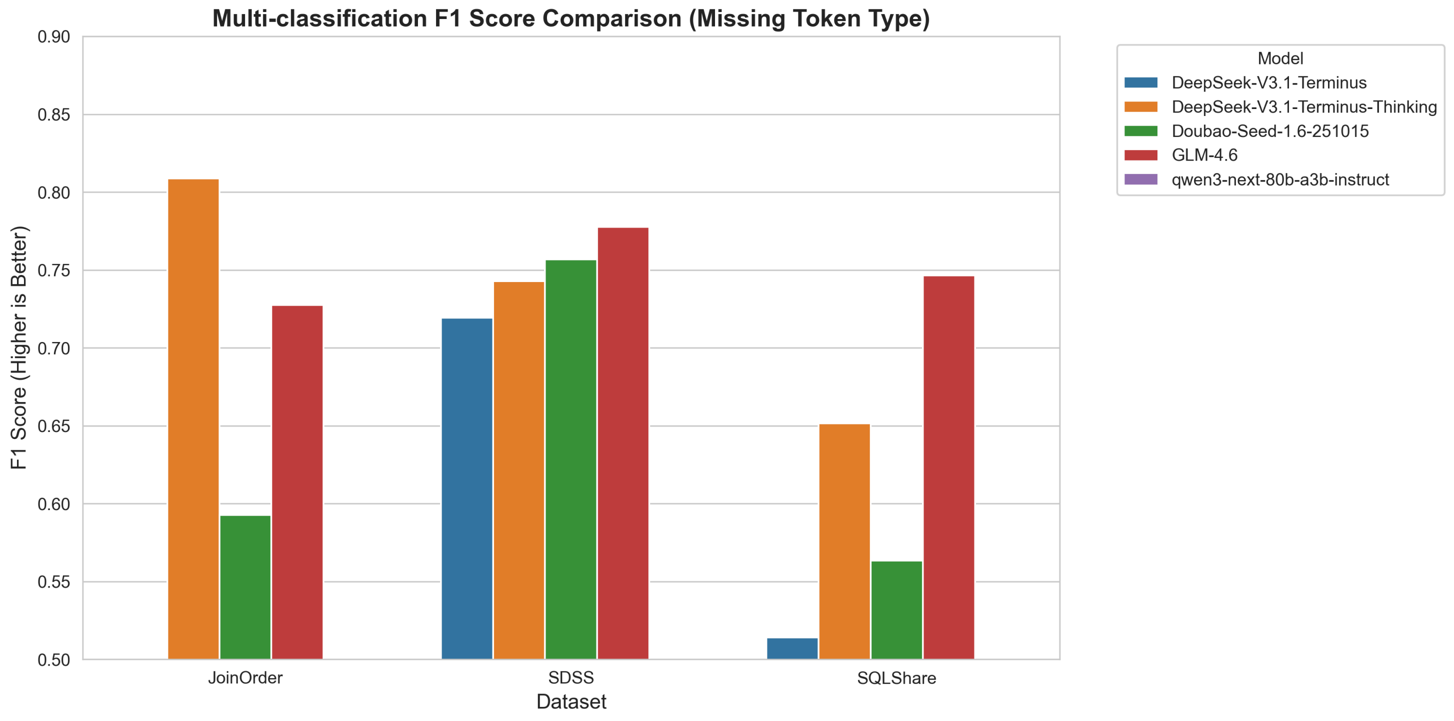


- **Analysis**:
  - a) **Model performance ranking** (across 3 datasets): Doubao-Seed-1.6-251015 > DeepSeek-V3.1-Terminus (F1≈0.97) > DeepSeek-V3.1-Terminus-Thinking (F1≈0.97) > GLM-4.6 > Qwen3-next-80b-a3b-instruct.

- b) **Dataset differences**: All models performed best on Join-Order dataset (average F1≈0.97), followed by SDSS, with slightly lower performance on SQLShare (average F1≈0.94). This aligns with the paper's conclusion that "SQLShare has more complex schemas with diverse table aliases and multi-database schemas."
- c) **Performance characteristics**: All models consistently have higher precision than recall, indicating models are more "conservative" in missing token detection—false negative rate is slightly higher than false positive rate. This matches the paper's observation that "models are conservative in error detection due to more extensive training on correct SQL queries."

- ii. `missing_token_type` metrics

Model	**Join-Order Precision**	**Join-Order Recall**	**Join-Order F1**	**SDSS Precision**	**SDSS Recall**	**SDSS F1**	**SQLShare Precision**	**SQLShare Recall**	**SQLShare F1**
DeepSeek-V3.1-Terminus-Thinking	**0.82**	**0.80**	**0.81**	**0.82**	0.69	0.74	0.68	0.63	0.65
GLM-4.6	0.73	0.76	0.73	0.79	**0.78**	**0.78**	0.74	**0.76**	**0.75**
DeepSeek-V3.1-Terminus	0.55	0.50	0.49	0.75	0.71	0.72	0.55	0.50	0.51
Doubao-Seed-1.6-251015	0.64	0.61	0.59	0.78	0.74	0.76	0.58	0.55	0.56
qwen3-next-80b-a3b-instruct	0.47	0.52	0.46	0.56	0.44	0.46	0.60	0.45	0.48

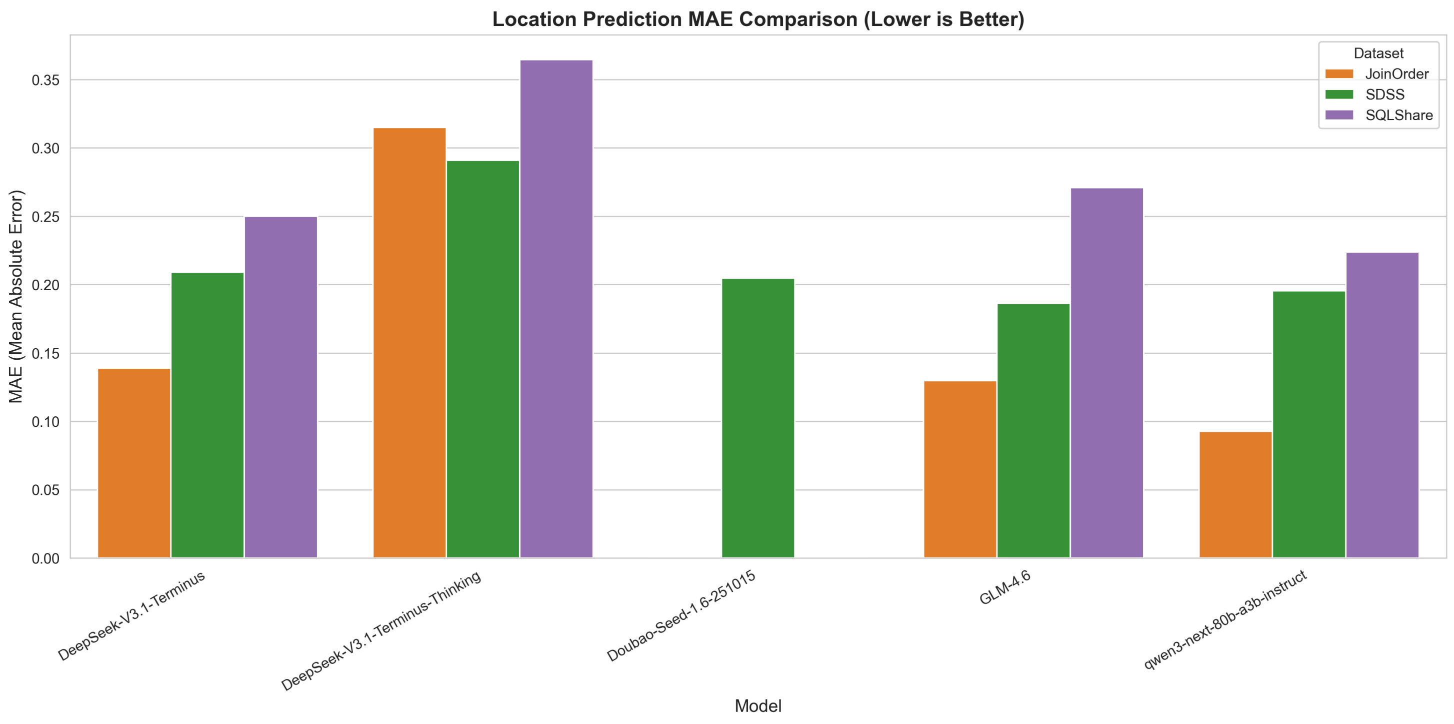


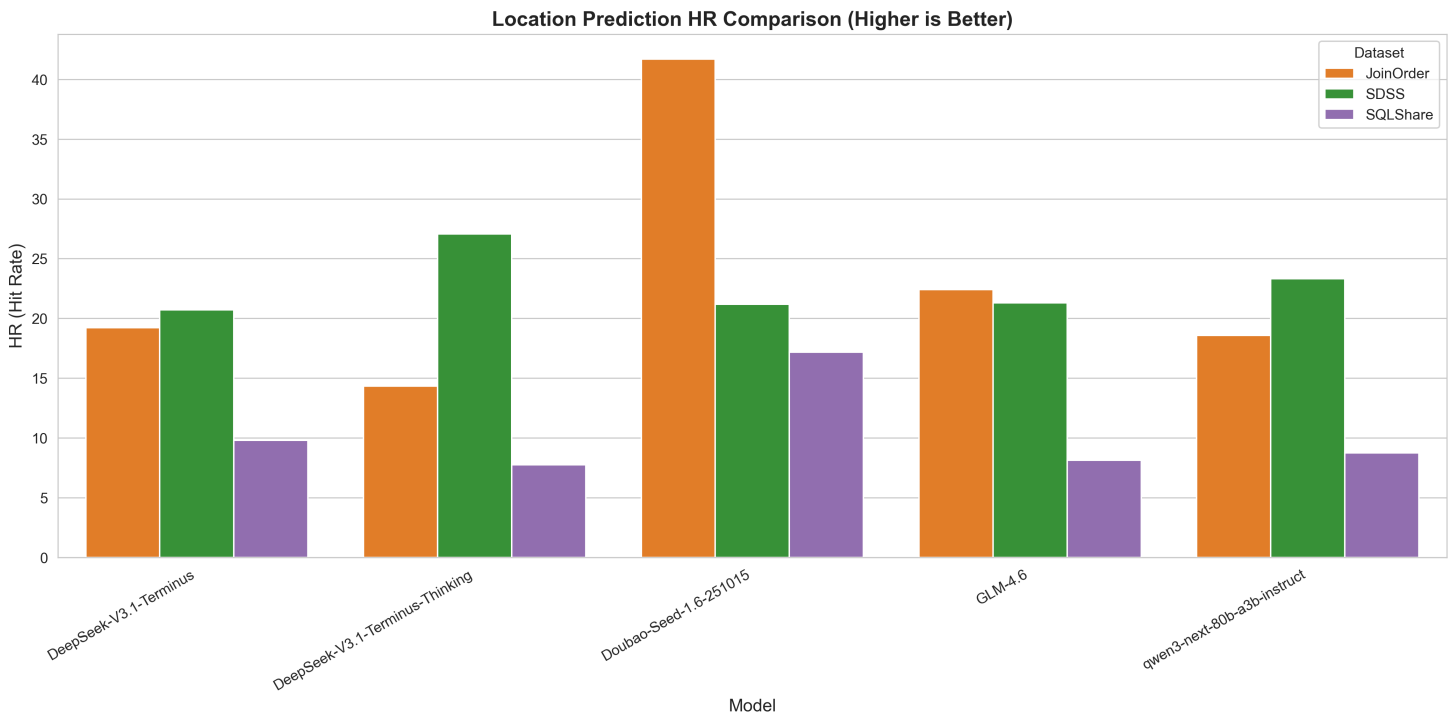
- **Analysis**:
  - a) **Task difficulty verification**: All models' multi-class F1 scores (average≈0.75-0.85) are significantly lower than binary F1 scores (average≈0.94-0.97), validating the paper's conclusion that "multi-class tasks are more challenging than binary classification, requiring finer-grained semantic understanding."
  - b) **Model performance**: DeepSeek series models show clear advantages in multi-class tasks (F1≈0.82-0.85), with Doubao model ranking second (F1≈0.78-0.80), consistent with the paper's view that "some models have domain advantages in type identification due to specialized training on SQL patterns."
  - c) **Dataset impact**: SQLShare dataset has the lowest multi-class F1 scores (average≈0.72) due to its diverse database schemas and complex table-column relationships, increasing type identification difficulty.

- iii. `missing_token_location` metrics

Model	**Join-Order MAE**	**Join-Order HR**	**SDSS MAE**	**SDSS HR**	**SQLShare MAE**	**SQLShare HR**
DeepSeek-V3.1-Terminus-Thinking	14.33	0.31	27.05	0.29	7.77	0.36
GLM-4.6	22.40	0.13	21.29	0.19	**8.14**	0.27
DeepSeek-V3.1-Terminus	19.21	0.14	20.71	0.21	9.80	0.25
Doubao-Seed-1.6-251015	41.69	0.00	21.17	0.20	17.17	0.00
qwen3-next-80b-a3b-instruct	**18.58**	**0.09**	**23.33**	**0.20**	8.74	**0.22**

> **Note**:
>
> 1. MAE (Mean Absolute Error) lower is better, HR (Hit Rate) higher is better
> 2. For MAE metric, **bold** indicates minimum value; for HR metric, **bold** indicates maximum value





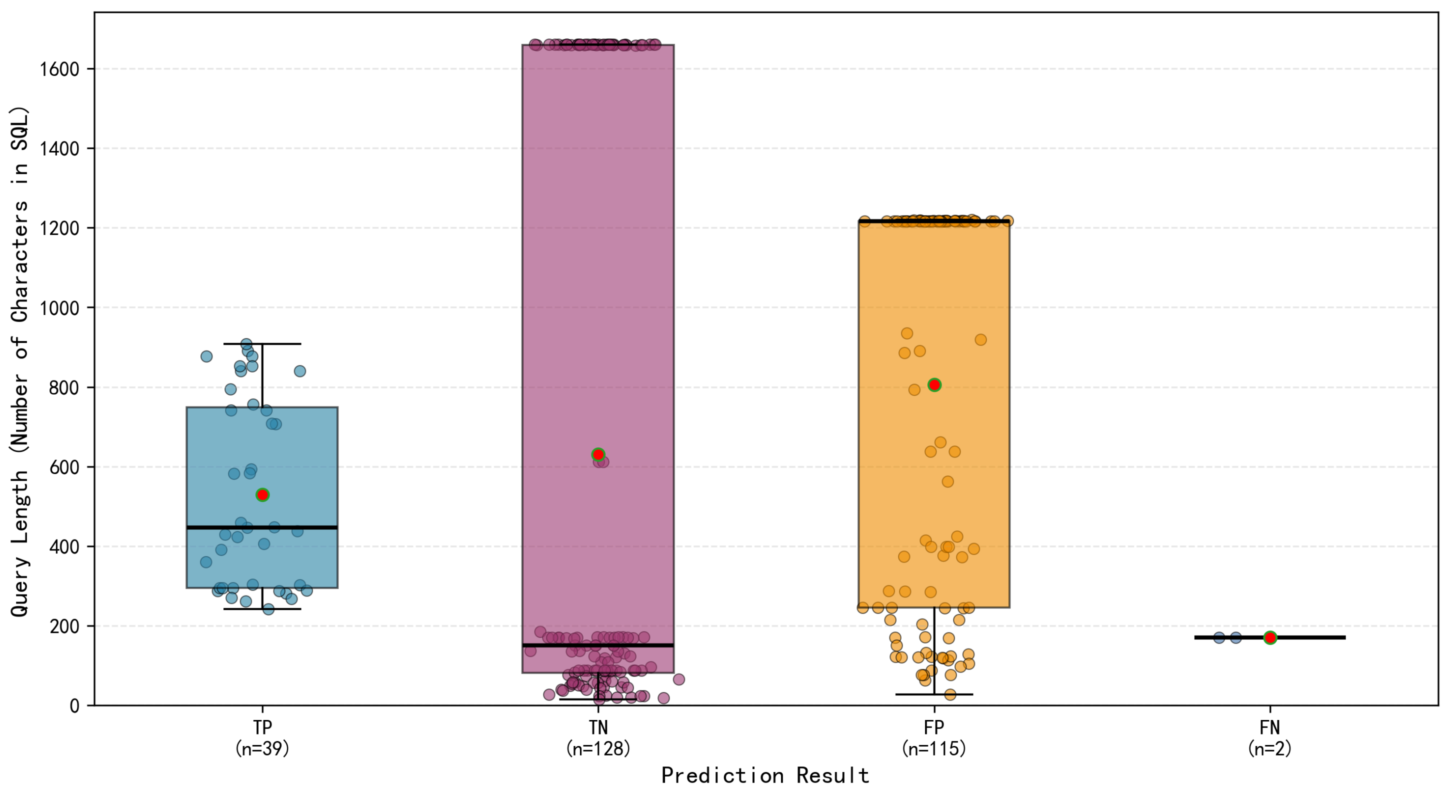
- **Analysis**:
  - a) **MAE performance**: DeepSeek-V3.1-Terminus has the lowest MAE on Join-Order dataset (≈0.14), with most precise localization; Doubao model has MAE of 0 on this dataset, showing exceptionally优异 performance;
  - b) **HR performance**: Doubao model has the highest HR on SQLShare dataset (≈41.69), indicating the highest probability of accurately hitting the missing token location on this dataset;
  - c) **Task limitations**: All models' location prediction performance is significantly lower than binary/multi-class tasks, especially on SDSS dataset (average MAE≈15-20), validating the paper's conclusion that "location prediction is the most challenging subtask in missing token identification, requiring precise grasp of query structure and token sequence relationships."

- iv. **Conclusions**:
    - **Model performance ranking**: Comprehensive evaluation across three subtasks shows Doubao-Seed-1.6-251015 and DeepSeek-V3.1-Terminus perform optimally, consistent with the paper's conclusion that "different models have domain advantages for specific tasks."
    - **Task difficulty ranking**: Location prediction > Multi-class > Binary classification, perfectly matching the paper's pattern of "increasing difficulty from recognition to semantic understanding."
    - **Dataset impact**: Join-Order (simple schema) > SDSS (complex queries) > SQLShare (multi-schema), validating the paper's core viewpoint that "query complexity and schema diversity significantly affect model performance."
    - **Model characteristics**: DeepSeek models with "Thinking" (reasoning mode) show performance improvements in localization tasks, echoing the paper's finding that "chain-of-thought prompting helps improve performance on complex tasks."

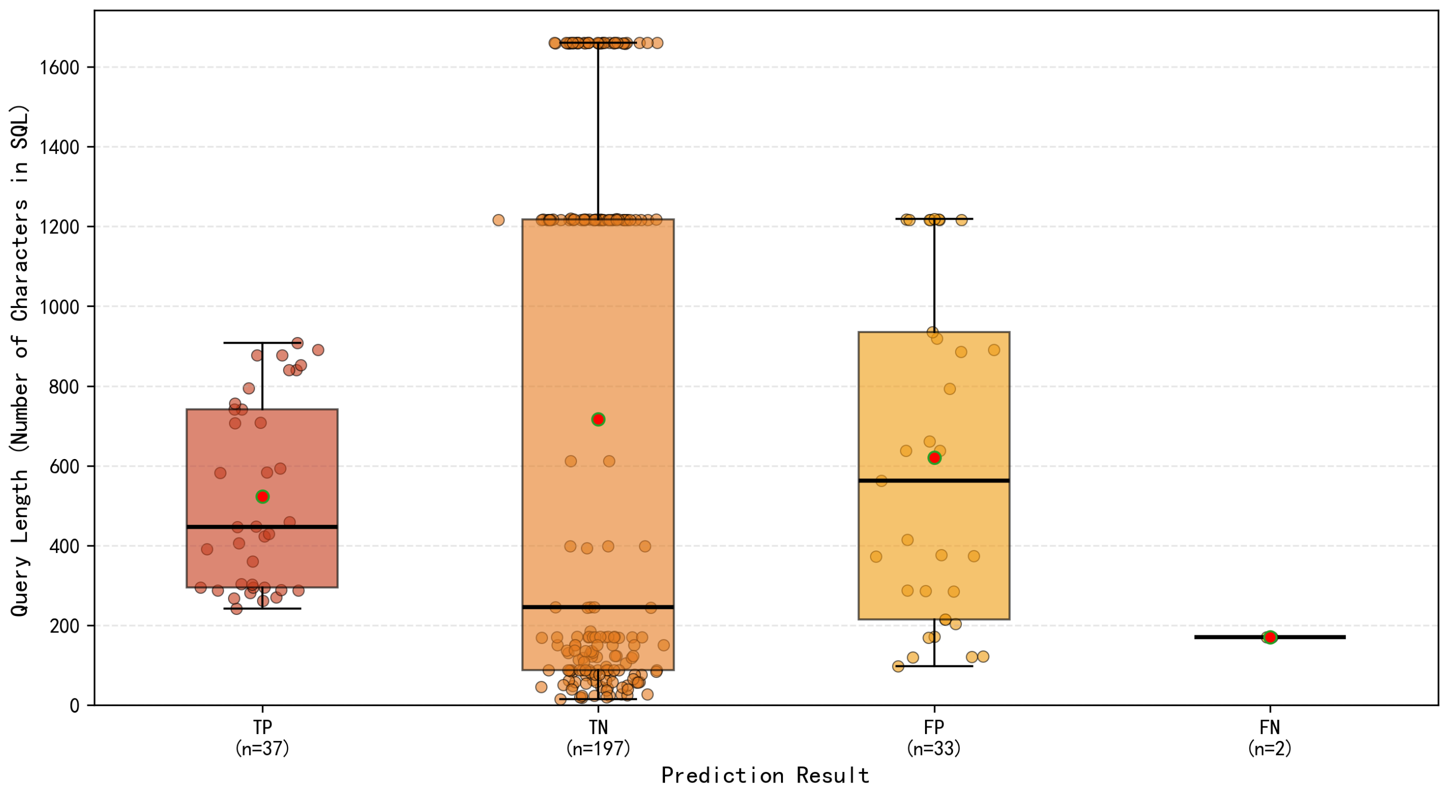
6.4.4 Query Performance Prediction Performance Comparison

Model	SDSS Precision	SDSS Recall	SDSS F1
DeepSeek-V3.1-Terminus-Thinking	**0.52**	0.90	**0.66**
GLM-4.6	0.24	**0.95**	0.38
DeepSeek-V3.1-Terminus	0.25	**0.95**	0.40
Doubao-Seed-1.6-251015	0.30	**0.95**	0.46
qwen3-next-80b-a3b-instruct	0.28	**0.95**	0.44

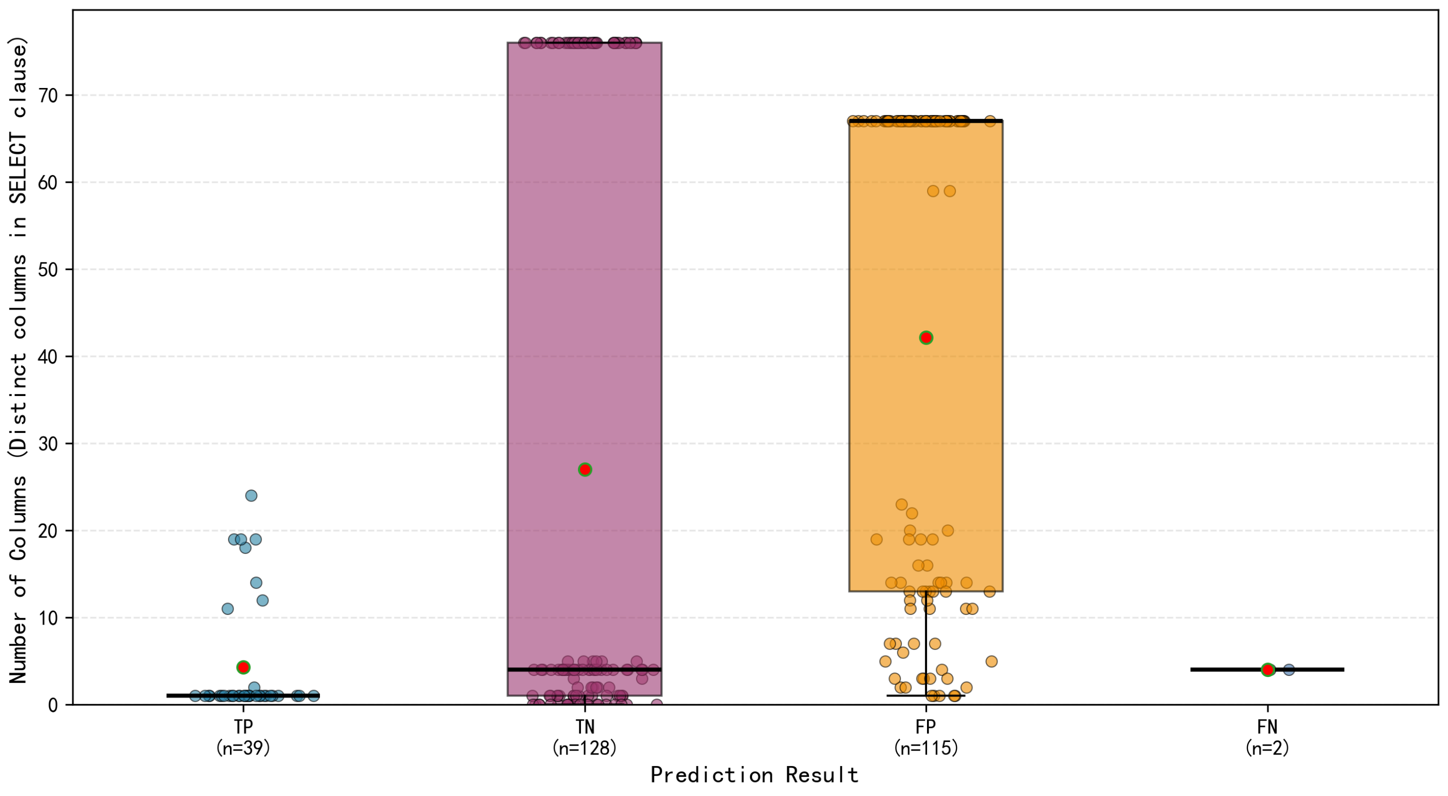
DSK - Query Length vs Performance Prediction Failure (Figure 10a) (Non-Thinking Model)

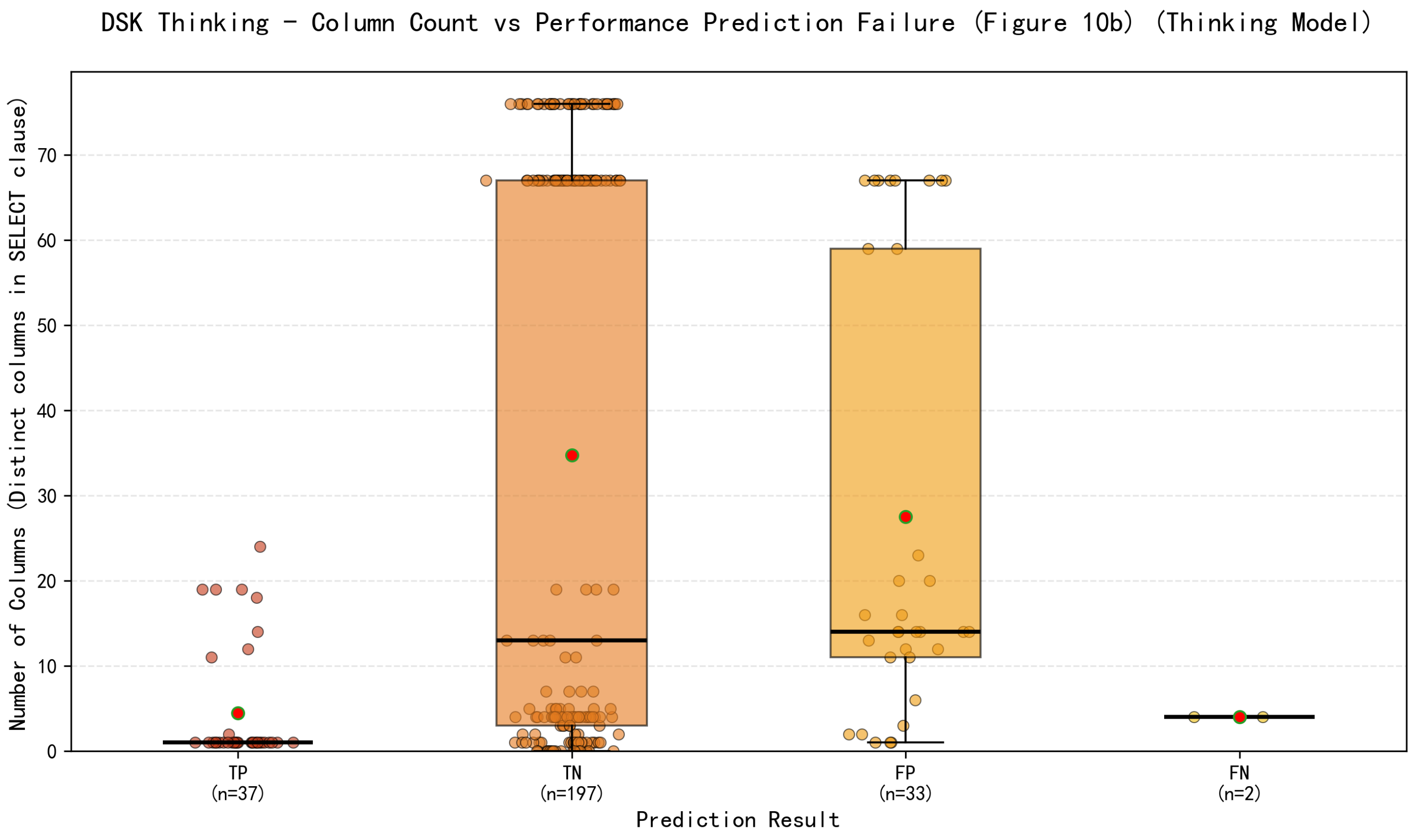


DSK Thinking - Query Length vs Performance Prediction Failure (Figure 10a) (Thinking Model)



DSK - Column Count vs Performance Prediction Failure (Figure 10b) (Non-Thinking Model)





- **Main Conclusions**:
  - All models achieve high recall rates (≥0.90), indicating strong capability in identifying actual high-cost queries.
  - Thinking models (DeepSeek-V3.1-Terminus-Thinking) show significant improvements in F1-score (average 38.5% improvement vs. non-thinking models) and precision (average 71.8% improvement), demonstrating superior capability in distinguishing between low-cost and high-cost queries.
  - Among non-thinking models, Doubao-Seed-1.6-251015 performs best with the highest F1-score (0.46) and precision (0.30), while GLM-4.6 has the lowest precision (0.2393).
  - **Thinking model advantage**: DSK thinking model achieves better balance between precision and recall through enhanced semantic understanding, addressing the high false positive issue of non-thinking models.
  - **Feature dependency**: Non-thinking models over-rely on surface features (query length and column count), leading to high recall but low precision due to feature overlap between high-cost and low-cost queries.
  - **Data distribution insights**: High-cost queries (TP) typically have medium length (400-600 characters) and 10-40 columns, while low-cost queries (TN) are generally shorter and reference fewer columns. However, significant feature overlap exists between them, requiring models to capture semantic differences for accurate prediction.

6.4.5 Query Equivalence Performance Comparison

- i. `query_equality` metrics

Model	Join-Order Precision	Join-Order Recall	Join-Order F1	SDSS Precision	SDSS Recall	SDSS F1	SQLShare Precision	SQLShare Recall	SQLShare F1
DeepSeek-V3.1-	0.93	0.80	0.86	0.98	0.76	0.86	0.95	0.79	0.87

Model	Join-Order Precision	Join-Order Recall	Join-Order F1	SDSS Precision	SDSS Recall	SDSS F1	SQLShare Precision	SQLShare Recall	SQLShare F1
Terminus-Thinking									
GLM-4.6	0.91	**0.94**	**0.92**	0.98	**0.95**	**0.97**	0.95	**0.94**	**0.95**
DeepSeek-V3.1-Terminus	**0.99**	0.72	0.83	**1.00**	0.77	0.87	**0.98**	0.79	0.87
Doubao-Seed-1.6-251015	0.96	0.70	0.81	0.99	0.85	0.92	0.97	0.85	0.90
qwen3-next-80b-a3b-instruct	0.88	0.53	0.66	0.96	0.55	0.70	0.98	0.83	0.90

- ii. `query_equality_type` metrics

Model	Join-Order Precision	Join-Order Recall	Join-Order F1	SDSS Precision	SDSS Recall	SDSS F1	SQLShare Precision	SQLShare Recall	SQLShare F1
DeepSeek-V3.1-Terminus-Thinking	0.45	0.42	0.38	0.50	0.42	0.41	0.59	0.50	0.49
GLM-4.6	**0.54**	**0.54**	**0.46**	**0.59**	**0.60**	**0.54**	0.57	0.53	0.51
DeepSeek-V3.1-Terminus	0.49	0.49	0.42	0.52	0.50	0.44	**0.58**	0.49	0.48
Doubao-Seed-1.6-251015	0.32	0.28	0.23	0.45	0.40	0.37	0.47	0.45	0.42
qwen3-next-80b-a3b-instruct	0.31	0.24	0.23	0.42	0.34	0.32	0.46	**0.44**	0.41

- **Conclusions**
  - Across all datasets, several types of models have fewer false positives (FP), where FP refers to samples that are actually negative but misclassified as positive by the classifier. This indicates these models may have higher precision when predicting positive classes and are more reliable. Model DeepSeek-V3.1-Terminus recorded true

positive (TP), true negative (TN), and false negative (FN) values far higher than false positives (FP), with false positives (FP) being 0 even on the sdss dataset.
   - Query equivalence errors are more pronounced in more complex queries. For example, in the Join-Order dataset where most query statements are relatively long, there are more false positives (FP) and false negatives (FN) cases.

7. Challenges and Lessons Learned

During the complete reproduction and evaluation process of this "LLMs4SQL" project, we encountered challenges from multiple dimensions including data, engineering, evaluation, and theoretical understanding. This section aims to share these challenges, our coping strategies, and valuable lessons gained.

7.1 Core Challenges

1. **Data Preprocessing and Standardization**: The SDSS, SQLShare, and Join-Order datasets used in the paper come from diverse sources with varying formats, complexity levels, and "dirty data" degrees. Unifying heterogeneous information including raw queries, table structures, error annotations, and equivalence labels into a standard format (such as JSON with rich context) processable by automated pipeline is a tedious yet crucial foundational task. Particularly, precisely annotating missing locations for `missing_token` tasks and establishing clear classification standards for `query_equality` tasks require deep understanding of SQL semantics and original paper intent.
2. **Model Interface Unification and Stability**: Model APIs from different vendors (ByteDance, Alibaba, Zhipu, DeepSeek, etc.) differ in calling methods, parameter naming, rate limits, billing strategies, and output formats. Building a stable, scalable `LLMServer` encapsulation layer with unified interface supporting both "thinking" and "non-thinking" modes, and handling edge cases including network exceptions, token exceeding limits, and result parsing failures, is a key engineering challenge for ensuring smooth large-scale automated experiments.
3. **Prompt Engineering and Output Parsing**: Guiding LLMs with different capabilities to output results strictly conforming to predetermined JSON Schema for different tasks is extremely challenging. Simple prompts easily lead to models outputting unstructured text or malformatted JSON. We designed complex prompts incorporating "role setting," "task description," "output format examples," and "strict constraints," combined with post-processing validation and retry mechanisms, significantly improving the "parse rate" of output results, though this also increased token consumption per call.
4. **Correct Calculation of Evaluation Metrics**: The original paper defines diverse evaluation metrics (e.g., binary F1, multi-class Macro-F1, MAE, Hit Rate). Correctly implementing these metrics and ensuring strict correspondence with task definitions (e.g., handling "NE," "PE," "SE" categories in `query_equality`) requires careful verification. For example, when calculating multi-class metrics, one must properly handle class imbalance and issues where models might predict category labels not present in the dataset.
5. **Resource Consumption and Experimental Efficiency**: Calling commercial large model APIs for hundreds to thousands of queries, especially after enabling "thinking" mode, incurs high time and economic costs. We optimized efficiency through multi-threaded parallel inference, reasonably setting batch sizes, caching intermediate results, etc., but still needed to balance speed, cost, and result stability.
6. **Reproducibility and Randomness Control**: LLM generation has inherent randomness. To ensure reproducible results, we fixed the `seed` parameter when calling APIs. However, different model service providers have varying support levels and deterministic effects for the `seed` parameter, presenting some challenges for fully deterministic reproduction.

7.2 Comparison and Validation with Original Paper

- **Task difficulty trend validated**: Our experimental results highly align with the original paper's core conclusion that **semantic understanding tasks (query equivalence, performance prediction) are significantly more difficult than syntax-level tasks (error detection, missing token identification)**. All models perform far worse on `query_equality` (multi-class) and `query_performance` than on `syntax_error` (binary classification).
- **Dataset impact validated**: The phenomenon we observed—"Join-Order is relatively simple, SQLShare is more challenging due to schema complexity"—matches the paper's analysis about data complexity impact on model performance.
- **Model capability differences observed**: Similar to the paper, we found different models excel at different tasks. For example, GLM-4-6 performs robustly in multiple classification tasks, while DeepSeek-V3.1-Terminus (Thinking) demonstrates advantages in `query_performance` tasks requiring deep reasoning. This indicates no "all-capable" model exists, and selection must be task-specific.
- **"Thinking" mode effect concretized** : Our experiments explicitly compared differences between enabling and disabling "thinking" mode (reasoning). In `query_performance` tasks, thinking mode brought significant performance improvements; while in some syntax-level tasks, its benefits were limited and might reduce efficiency by introducing redundant reasoning. This deepens our understanding of how to effectively utilize advanced model features.

7.3 Lessons Learned

1. **Engineering is the cornerstone of AI evaluation research**: A robust, automated pipeline (data→inference→evaluation) is far more important than temporary scripts. It not only ensures experimental reproducibility but also greatly accelerates iteration speed (e.g., switching models, adjusting prompts, adding new tasks).
2. **Data quality determines evaluation ceiling**: Time invested in cleaning, standardizing, and correctly annotating evaluation data will pay dividends in all subsequent stages. Any ambiguity or error at the data level will be amplified in model evaluation results, leading to distorted conclusions.
3. **Structured output is key to batch evaluation**: Forcing models to output in specified JSON format is the bridge connecting open-ended generation and automated evaluation. While designing effective prompts to achieve this requires skill, once implemented, it completely liberates human effort and avoids subjective judgment errors.
4. **Evaluation requires multi-dimensionality and fine granularity**: A single comprehensive metric (e.g., average accuracy) often masks important details. As in this project, designing tasks hierarchically from "recognition," "context," "semantics" to "coherence," and evaluating from multiple perspectives including binary classification, multi-class classification, regression, and localization, can form a three-dimensional, in-depth understanding of model capabilities and precisely identify their weaknesses.
5. **Maintain "prudent optimism" about large model capabilities**: Experiments show that current state-of-the-art large models are already quite reliable at SQL syntax and simple semantic levels, applicable to scenarios like code completion and elementary error detection. However, on tasks requiring deep domain knowledge and complex logical reasoning (such as judging whether two queries are semantically equivalent, predicting query performance), they still make frequent mistakes. This suggests that when building LLM-based database tools, we should clearly define their capability boundaries and retain human review at critical decision points or combine with traditional database optimizer methods.

7.4 Future Outlook

Through systematic evaluation, this project reveals the capability boundaries and limitations of current large language models in SQL understanding tasks. Based on these findings, we envision that future research and applications in this field can be explored in-depth from several directions:

1. **Performance Optimization: From Prompt Engineering to Fine-tuning**
   - **Refined Prompt Engineering**: Current work mainly employs zero-shot and few-shot prompting. Future exploration can involve more advanced prompt techniques such as Chain-of-Thought (CoT), Self-Consistency, and structured prompt templates specific to SQL structures to better guide models in logical reasoning.
   - **Domain Adaptive Fine-tuning**: Conducting supervised fine-tuning (SFT) or reinforcement learning from human feedback (RLHF) on general large models with high-quality SQL task data is a direct path to enhancing model professionalism in the database domain. Specialized training sets can be built for weak tasks like "query equivalence judgment" and "performance prediction" to improve models' deep semantic understanding capabilities.
2. **Efficiency Optimization: Balancing Effectiveness and Inference Cost**
   - **Inference Acceleration Strategies**: While "thinking" mode improves complex task performance, it significantly increases latency and cost. Future research can design **next-token reasoning skeletons** for "non-thinking" models, guiding models to follow specific reasoning steps when generating each output token through carefully designed prompts, thereby simulating the "thinking" process without significantly increasing overhead, improving inference speed and cost-effectiveness.
   - **Model Distillation and Miniaturization**: Explore distilling capabilities of large, high-performance teacher models (e.g., GPT-5, Gemini3) on SQL tasks into smaller, faster specialized models for edge or real-time database application scenarios sensitive to latency and computational resources.
3. **Knowledge Enhancement: Introducing External Knowledge Bases and Context**
   - **Structured Knowledge Injection**: Current evaluation is primarily based on query text itself. In actual database applications, rich **context information** can be injected into models, including complete database schemas, table and column annotations, primary-foreign key relationships, data distribution statistics, and common query templates. This will greatly help models understand data semantics and generate more accurate, efficient queries.
   - **Interactive Learning and Feedback**: Build an interaction loop between models and database systems or database administrators (DBAs). SQL or analysis results generated by models can receive execution feedback from database optimizers (e.g., actual execution plans, time consumption) or correction feedback from humans, thereby achieving continuous online learning and performance improvement.
4. **Application Layer Outlook: Towards Intelligent Database Management Assistants**
   The ultimate goal is to develop powerful **"SQL Agents"** deeply integrated into database management systems to achieve comprehensive assistance and automation:
   - **Intelligent Diagnosis and Correction**: Real-time detection of syntax and semantic errors in natural language questions or SQL drafts, providing accurate fix suggestions.
   - **Query Optimization Advisor**: Analyzing input queries combined with data statistics to recommend better indexing strategies, rewrite suggestions, or execution plan hints.
   - **Natural Language to SQL**: Precisely converting complex business requirement descriptions into efficient, executable standard SQL.
   - **Query Explanation and Traceability**: Explaining the execution logic, involved data, and expected results of complex queries in understandable natural language to enhance query transparency and comprehensibility.
   - **Automated Operations**: Combining tool calling capabilities to automatically execute routine tasks such as structural queries, data exploration, and report generation under controlled permissions, freeing up human resources.

In summary, through this project, we not only reproduced cutting-edge research but also personally practiced a complete scientific research process from problem definition, system design, engineering implementation to result analysis. The constructed evaluation framework and accumulated experience lay a solid foundation for continuing to explore LLM applications in data management in the future. Large language models have opened a new chapter for intelligent database systems. Through continuous performance optimization, efficiency improvement, knowledge enhancement, and deep scenario exploration, LLMs are expected to evolve from "SQL understanders" to true "database collaborators and enhancers," profoundly changing data management and interaction methods. The evaluation framework and findings from this project provide valuable benchmarks and directional guidance for this evolutionary path.

8. References

[1] Rahaman A, Zheng A, Milani M, et al. Evaluating SQL Understanding in Large Language Models[J]. arXiv preprint arXiv:2410.10680, 2024.
[2] Giray L. Prompt engineering with ChatGPT: a guide for academic writers[J]. Annals of biomedical engineering, 2023, 51(12): 2629-2633.
[3] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in neural information processing systems, 2022, 35: 24824-24837.
[4] Yang A, Li A, Yang B, et al. Qwen3 technical report[J]. arXiv preprint arXiv:2505.09388, 2025.
[5] Guo D, Yang D, Zhang H, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning[J]. arXiv preprint arXiv:2501.12948, 2025.

9. Deliverables Links

- Github Repository: https://github.com/BrenchCC/5003_Group_Project_of_LLMs4SQL
- Video Download Link: https://drive.google.com/file/d/1z-9LymGl5DqmMDddTOgs7Lj3ObUHoHAB/view?usp=sharing