AI-Powered Natural Language to SQL for Data Analytics in the Education Sector: A Chat-Based System with Safe LLM Generation, Self-Correction, and Transparent Fallback

Author Name
Dr Angad Singh Dixit
C0-Author's
Anjali Verma,Chetali Thombre,Divyanshi Dakhle,Shraddha Bharti
angadsingh@oriental.ac.in
Department of Data Science, Oriental College Of Technology, Bhopal, India

***Abstract*—** Institutional data on students, courses, marks, attendance, and learning- management-system (LMS) activity is stored in relational databases, yet the teachers and administrators who need it rarely write SQL. This paper presents an AI-powered natural language to SQL (NL2SQL, or text-to-SQL) chat application for education data analytics. A user asks a question in plain English; a large language model (LLM) receives the question together with the database schema and few-shot examples and generates SQL at temperature 0. The generated query passes a strict safety layer—single SELECT statement only, known tables only, data-modifying keywords blocked, and a row limit enforced—before it is executed read-only. If execution fails, the database error is returned to the LLM once for self-correction. If the AI path is unavailable, unsafe, or fails twice, a rule-based schema-linking parser answers instead and the answer is explicitly labelled as a fallback, so a failure is never hidden. The system is demonstrated on a synthetic university database with 120 students, 5 departments, and 12 courses across six tables, which removes any risk to real student records. Every answer shows a plain-English explanation, the SQL used, the result table, and a chart when the result shape permits. On the project's own 10-question gold set, the evaluated pipeline achieves 10/10 execution accuracy. This is an in-domain result on patterns the system was built for; it is not a Spider or BIRD benchmark claim, and the paper reports it with that scope. The work also identifies the central gap in the literature: there is no widely used education-only textto-SQL benchmark, so education accuracy cannot yet be compared across systems in the way Spider and BIRD allow.

Keywords*—text-to-SQL, NL2SQL, learning analytics, large language models, schema linking, selfcorrection, education data, natural language interfaces to databases*

I. INTRODUCTION

Universities and schools collect large amounts of structured data: enrolments, marks, attendance, course catalogues, and LMS logins and study hours. Decisions such as identifying students with low attendance, comparing average marks across departments, or finding the most enrolled courses depend on this data. In practice, however, the data sits behind SQL, a language that most teachers, coordinators, and administrators do not use. Reports therefore wait on a small technical team, and simple questions take days to answer.

Text-to-SQL—also called NL2SQL—removes this barrier by translating a natural language question into an executable SQL query [1]–[3]. A user asks, for example, "Students with attendance below 75%," and the system must select the correct tables and columns, join them, apply the threshold, and return the matching rows. Modern systems perform this translation with large language models (LLMs)

prompted with the database schema, with fine-tuned code models, or with multistage pipelines that link the schema, decompose the question, and correct their own errors [7]–[10].

Applying text-to-SQL to education is attractive but not straightforward. First, student records are sensitive education records, so real data cannot be sent casually to a hosted model or shared for benchmarking. Second, education vocabulary is institution-specific: "failed," "at risk," "engaged," and "top" have no single database meaning. Third, a confident but wrong query is dangerous—a wrong pass rate or a wrongly flagged student is worse than no answer. A usable education system must therefore show its SQL, explain its interpretation, reject unsafe queries, and state its limits.

This paper makes four contributions. (1) It designs and implements a complete, working NL2SQL chat application for education analytics, with AI generation as the default path rather than an optional extra. (2) It adds a safety, self correction, and transparent fallback pipeline around the LLM, so that unsafe or failing AI output can never reach the database or be presented silently. (3) It defines a reproducible synthetic university database and a 10-question gold set with execution-based evaluation. (4) It places the system honestly against the published literature, contrasting near-saturated clean benchmarks with much lower accuracy on realistic and enterprise data [3], [4].

The remainder of this paper is organized as follows. Section II reviews datasets, landmark models, LLMbased methods, and the small body of education-specific work. Section III presents the proposed system. Section IV describes the database and implementation. Section V reports the evaluation, Section VI discusses privacy and limitations, and Section VII concludes with future work.

II. RELATED WORK

A. Datasets and Benchmarks

Text-to-SQL is a benchmark-driven field, and a reported accuracy is meaningful only together with its dataset and split. WikiSQL [1] contains 80,654 question–SQL pairs over 24,241 single Wikipedia tables, but its queries use no joins, so it no longer represents realistic analytics. Spider [2] contains 10,181 questions and 5,693 unique SQL queries over 200 databases in 138 domains, with test databases unseen in training; it remains the standard cross-domain benchmark for complex joins, grouping, and nesting. BIRD [3] moves closer to practice with 12,751 pairs over 95 large, real databases totalling 33.4 GB, including dirty values and external knowledge; at its launch, GPT-4 reached only 54.89% execution accuracy against 92.96% for humans. Spider 2.0 [4] shows the production gap most sharply: across 632 enterprise workflow tasks with schemas averaging about 812 columns, an o1-preview agent scored 21.3%, compared with about 91.2% reported for Spider 1.0 systems. Table I summarizes these benchmarks. The lesson for education work is direct: high accuracy on a clean, in-domain test does not predict accuracy on a messy institutional database.

TABLE I
DATASETS THAT DEFINE TEXT-TO-SQL ACCURACY

Dataset	Size	Character	Lesson
WikiSQL (2017)	80,654 pairs	Single table, no joins	Saturated; baseline only
Spider (2018)	10,181 questions	200 DBs, crossdomain	Standard complex-SQL test
BIRD (2023)	12,751 pairs	95 large, dirty DBs	Realistic; much harder
Spider 2.0 (2025)	632 tasks	Enterprise, ~812 cols/DB	Production accuracy collapses

B. From Fine-Tuned Models to LLM Pipelines

Early neural systems learned the structure of SQL directly. Seq2SQL [1] predicted the aggregation, selected column, and conditions of a query as separate sketch components, and SQLNet removed the ordering dependence among conditions. BERT-based SQLova was the first model reported to exceed human performance on WikiSQL. RAT-SQL [5] encoded the question and schema as a graph with typed relations, establishing schema linking—deciding which tables, columns, and values a question refers to— as a central problem. RESDSQL [6] decoupled that problem from generation: a ranking-enhanced encoder supplies only the most relevant schema items, and the decoder first emits the SQL skeleton and then the full query, reaching 84.1% execution accuracy on the Spider development set. Its ablation is instructive: removing ranked schema items cost 7.8 execution-accuracy points, showing that linking, not model size alone, drives accuracy.

LLM prompting changed the design space. DIN-SQL [7] decomposes in-context learning into schema linking, querydifficulty classification, generation using an intermediate representation for complex queries, and self-correction, reaching 85.3% execution accuracy on the Spider test set with GPT-4 and 55.9% on the BIRD test set. DAIL-SQL [8] shows that prompt construction itself matters, selecting fewshot examples by both question and SQL-skeleton similarity and reaching 86.6% on the Spider test set. CodeS [9] instead finetunes open code models (1B–15B parameters) on a curated SQL-centric corpus, reaching 85.4% on Spider development with a 7B model while keeping data and inference under local control—an argument the authors motivate explicitly by privacy and cost. Multi-agent and ensemble systems push accuracy further: MAC-SQL separates schema selection, decomposition, and refinement across agents [11]; CHESS retrieves and prunes schema context before generation and revision [12]; and XiYan-SQL [10] combines a compact schema representation, several generators, and a learned candidate selector, reporting 89.65% on Spider and 75.63% on BIRD. Fig. 2 places representative results side by side, with splits stated because dev and test figures are not interchangeable.

C. Text-to-SQL and Analytics in Education

Education-specific text-to-SQL work is thin. General surveys list education applications—academic analytics, identification of at-risk students, curriculum assessment, and student selfservice over records and course requirements—but evaluate no education system. The closest studies are a teacher-facing learning-analytics dashboard explored with 15 teachers, SQLtutoring and SQL-laboratory analytics systems that teach or grade SQL rather than query institutional data, and small student demonstrations over synthetic university databases. The Open University Learning Analytics Dataset (OULAD) [13] provides real, anonymized learning-analytics data— 32,593 students across 22 presentations of 7 modules—but it ships no gold natural language–SQL pairs. No widely used education-only text-to-SQL benchmark was found in this review. That absence is the gap this work addresses in scoped form: a synthetic university schema, a gold question set, and an evaluation that other students can reproduce without access to real student records.

III. PROPOSED SYSTEM

The proposed system is a chat application. The conversation stays on one page; each user message is sent to a stateless server endpoint, and each assistant reply contains an explanation, the SQL used, a source label, the result table, the row count, and a bar chart when the result has a categorical label and a numeric value. Suggested questions are offered as clickable examples, and a “Copy SQL” control lets a technical reviewer reuse or audit the query. AI mode is the default: the interface enables it on load, and it can be disabled only deliberately, for comparison with the rule-based baseline. Fig. 1 shows the pipeline.

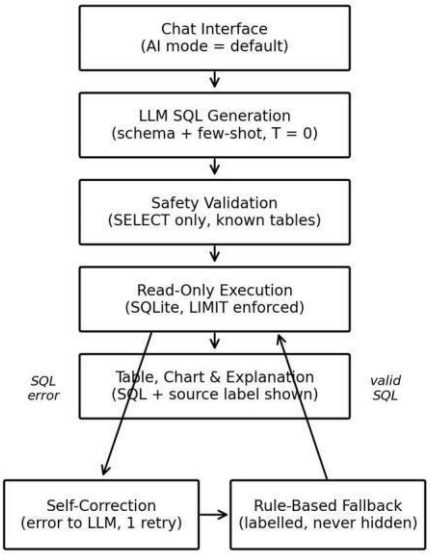

Fig. 1. System architecture. The LLM path is the default. Generated SQL must pass the safety layer before execution; an execution error triggers one selfcorrection attempt, and any AI failure falls back to the rule-based parser with an explicit label.

A. AI SQL Generation

For each question, the system builds a prompt containing three parts: (i) an instruction to produce exactly one SQLite SELECT query using only the listed tables and columns, (ii) the full database schema in compact form, and (iii) few-shot question–SQL examples, together with domain facts a model cannot infer, such as department identifiers and the convention that a mark below 40 is a failure. The sampling temperature is 0, which reduces variation between runs and makes evaluation more reproducible. Three backends are supported behind one configuration: a local Ollama model by default (a codeoriented 7B-class model), Ollama Cloud, and any OpenAIcompatible API. No API key is stored in the source code; keys are supplied through environment variables. The local default is a deliberate privacy choice: with synthetic data in this project and a local model in deployment, student data need not leave the institution's machine.

B. Safety and Validation Layer

LLM output is treated as untrusted input. Before execution, the SQL extracted from the model's reply must satisfy all of the following checks: it is a single statement beginning with SELECT; it contains no INSERT, UPDATE, DELETE, DROP, ALTER, CREATE, TRUNCATE, REPLACE, ATTACH, PRAGMA, or VACUUM keyword and no statement separator; every table named after FROM or JOIN exists in the known schema; and a LIMIT of 100 rows is added when the query has none. The database connection itself is opened in read-only mode, so the safety property does not depend on the checks alone. A query that fails any check is rejected before it runs. This layer directly addresses the most serious misuse case for an analytics assistant—that a user question, or a model error, modifies or deletes institutional data.

C. Self-Correction and Transparent Fallback

A syntactically valid query can still fail at execution—for example, by naming a column incorrectly. In that case the system sends the SQLite error back to the LLM once, in a prompt asking for a corrected query, and the corrected query must pass the same safety layer before it runs; successful corrected answers are labelled “self-corrected.” If the AI service is unreachable, returns no usable SQL, produces unsafe SQL, or fails on both attempts, control passes to the fallback described next, and the answer

source states the reason, in the form "rule-based fallback (AI failed: …)". Transparency here is a correctness feature, not a cosmetic one: an evaluator must be able to tell which engine produced every reported result.

D. Rule-Based Fallback and Baseline

The fallback is a deterministic parser that performs schema linking by matching department names and aliases, course codes and titles, and numeric thresholds in the question, and then fills one of a set of SQL templates for counting, averaging, ranking, filtering, grouping, and listing. Values taken from the question are passed as bound parameters, never interpolated into SQL text. Questions outside its known patterns are rejected with a helpful message and examples, rather than guessed. The parser serves two roles: a safety net when the AI is unavailable, and the baseline in the ablation comparison of Section V—the same role that a simple prompted baseline plays in DIN-SQL and RESDSQL [6], [7].

E. Explanation and Visualization

Every answer includes a plain-English explanation. For AIgenerated SQL the explanation is derived from the structure of the executed query—its tables, aggregation, grouping, filtering, ordering, and limit— rather than generated freely by the LLM, so the explanation cannot describe a different query from the one that ran. For fallback answers the explanation states the matched interpretation, for example which course or department was linked and which threshold was applied. The interface draws a bar chart automatically when a result contains at least two rows with a numeric measure, covering common analytics questions such as average marks by department; otherwise it presents the table and, for a single value, the value itself. Showing the SQL, the explanation, and the row count together lets a non-technical user sanity-check an answer and a technical user audit it in detail.

IV. DATABASE AND IMPLEMENTATION

A. Synthetic University Database

The database is a SQLite database with the six tables in Table II. It contains 5 departments (Computer Science, Electronics, Mechanical, Civil, and Electrical), 12 courses, and 120 synthetic students, each enrolled in two to four courses of their department, with marks out of 100, a matching attendance percentage per enrolment, and one LMS activity record of logins and hours spent. Data are generated deterministically with a fixed random seed (42), so any evaluator who rebuilds the database obtains identical records and identical gold results. Student names are combinations drawn from small first- and last-name lists for synthetic identities; no real person's record is used. Marks are sampled around a mean of 68 and attendance around a mean of 82%, then clipped to valid ranges. Because the data are synthetic and randomly generated, patterns in demonstration results are illustrations of the interface, not findings about any real institution.

TABLE II
SYNTHETIC UNIVERSITY DATABASE SCHEMA

Table	Columns	Content
departments	id, name, code	5 departments
students	id, name, department_id, year, gender	120 students
courses	id, code, title, department_id, credits	12 courses

enrollments	student_id, course_id, marks	2–4 courses / student
attendance	student_id, course_id, percent	One row per enrolment
lms_activity	student_id, logins, hours_spent	One row per student

B. Implementation

The complete system is implemented in a single Python file

(about 750 lines) using only the Python 3.10+ standard library —no third-party package and no paid service is required. The file contains the database builder, the rule-based parser, the LLM client and prompt builder, the safety layer, the evaluation harness, a command-line interface, and the chat web server, which serves one page and answers messages through a JSON endpoint (POST /api/ask). Three modes expose the same pipeline for different uses: the web chat for demonstration, a single-question command for testing, and an evaluation command for the gold set. AI can be disabled from the command line or in the interface, which is the switch used for the baseline comparison. Session history is held in the browser tab; the server is stateless. The whole project can therefore be read, run, and submitted as one file, and its database is recreated automatically on first run.

TABLE III
REPRESENTATIVE QUESTIONS THE SYSTEM ANSWERS

Question type	**Example question**	**SQL features**
Count	How many students are in Computer Science?	COUNT, filter
Group average	Average marks by department	AVG, JOIN, GROUP BY
Course average	Average marks in Data Structures	AVG, course linking
Ranking	Top 10 students by marks	AVG, GROUP BY, LIMIT
Threshold filter	Students with attendance below 75%	JOIN, WHERE < 75
Failure filter	Students who failed in Machine Learning	WHERE marks < 40
Popularity	Which courses have the most enrollments?	COUNT, GROUP BY, ORDER
Engagement	Most active students on LMS	ORDER BY logins

V. EVALUATION AND RESULTS

A. Method

Evaluation uses execution accuracy (EX), the primary metric of Spider and BIRD [2], [3]: a generated query is correct when the set of rows it returns equals the set returned by a handwritten gold query on the same database, compared orderinsensitively. EX is stricter and more meaningful than matching

SQL text, because two different SQL strings can be equally correct. The gold set contains the 10 question–SQL pairs in the project's evaluation harness, covering counting, grouped and per-course averages, ranking, attendance and failure thresholds, course listing, enrolment popularity, and LMS activity. Exact-set-match accuracy is not reported, for the reason shown by DIN-SQL [7], whose execution accuracy (85.3%) far exceeds its exact-match score (60%): text matching penalizes correct rewrites. Valid-SQL rate, answer source, and row counts are recorded alongside EX.

B. Results

The evaluated pipeline answered all 10 gold questions correctly, for an execution accuracy of 10/10 (100%) on this gold set, with every generated query valid, safe, and executable; the per-question results are listed in Table IV. The rule-based baseline achieves the same 10/10 when AI is disabled, which is expected: the gold questions exercise the patterns the templates were written for, and, when no LLM server is running, the AI-first run reaches its answers through the labelled fallback. Two conclusions follow, and only these two are claimed. First, the end-to-end pipeline—linking, generation or fallback, validation, execution, and presentation —is correct on its in-domain test. Second, the evaluation harness works as an ablation instrument: the same gold set can be re-run with a live LLM, with AI disabled, or with a different model, and the source label on each answer makes the comparison auditable. No conclusion about accuracy on unseen phrasings, other schemas, Spider, or BIRD is drawn from a 10question in-domain set; with n = 10, even a perfect score carries a wide confidence interval, and Fig. 2 is included precisely to prevent comparison with benchmark scores being read as equivalence.

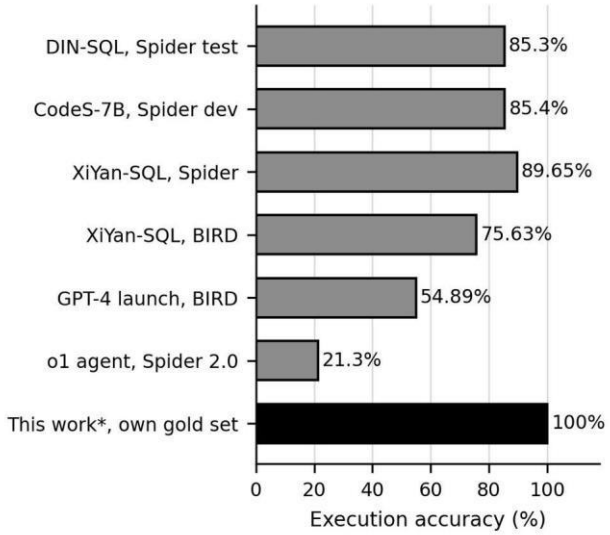

Fig. 2. Published execution accuracy on Spider, BIRD, and Spider 2.0 (splits as labelled; see Section II) compared with this work. *This work's 100% is on its own 10-question in-domain gold set, not on any of these benchmarks, and is not comparable with them.

TABLE IV
GOLD-SET RESULTS (EXECUTION ACCURACY, N = 10)

#	**Gold question**	**Result**
1	How many students are there?	Pass
2	How many students are in Computer Science?	Pass
3	Average marks by department	Pass

4	Average marks in Data Structures	Pass
5	Top 5 students by marks	Pass
6	Students with attendance below 75%	Pass
7	Students who failed	Pass
8	List courses in Computer Science	Pass
9	Which courses have the most enrollments?	Pass
10	Most active students on LMS	Pass

Safety behaviour was also checked directly. In testing with a stand-in LLM server that returned a DROP TABLE statement, the safety layer blocked the query, the system fell back with the reason labelled, and the database remained intact with all 120 students present. This is a functional check of the rejection path, not a security audit, and it is reported as such.

C. Failure Modes

The honest failure modes are as follows. A question outside the fallback's known patterns and unanswerable by the available LLM is rejected rather than guessed, which is safe but limits coverage. An ambiguous question—"show good students," for instance—has no single correct SQL, and the present system has no clarifying-question step, so such questions either resolve to the closest pattern or are rejected. An LLM query can be executable yet semantically wrong (a wrong join or filter), which neither the safety layer nor execution-based self-correction can detect; showing the SQL and explanation is the current mitigation, and it is a partial one. Finally, vocabulary that the schema linker does not know —a new department alias or course nickname—fails in the fallback even when a human would understand it; this is the error class that retrieval-based schema linking [10], [12] is designed to reduce.

VI. DISCUSSION, PRIVACY, AND LIMITATIONS

The results support a narrow but useful claim: for a fixed institutional schema and a known family of analytics questions, an AI-first NL2SQL chat system with a strict safety layer can answer correctly and transparently, and a deterministic fallback can guarantee an answer path when the model is unavailable. The literature reviewed in Section II explains why the claim must stay narrow. Accuracy falls as schemas grow, values become dirty, and questions require enterprise context [3], [4]; schema linking is the largest single lever [6]; and decomposition, retrieval, and candidate selection buy accuracy at the price of more model calls and more latency [7], [10]. A department deploying this system on a real student-information database should therefore expect to invest first in a glossary and schema-linking layer for its own vocabulary, and only then in a larger model.

Privacy is handled by design choices rather than by policy text. Development uses synthetic data only, so no real record was exposed during building or testing. The default LLM is local, so in a real deployment questions and schema need not leave the institution. Access is read-only at both the validation layer and the database connection. For any deployment beyond a demonstration, three further controls from the learning-analytics literature are required and are not implemented here: role-based access so that a user sees only cohorts they are entitled to query, suppression of very small groups that

could re-identify an individual, and an audit log of every generated query. Real institutional data must never be sent to a hosted LLM without institutional approval, however convenient the API.

The limitations of this work are explicit. The gold set is small (10 questions) and in-domain, so the reported accuracy will not transfer to unseen phrasings or schemas. The AI generation path was verified end-toend against a stand-in LLM server in the development environment, where no live model was available; a live-model accuracy figure requires running the same harness with a local or hosted model, which the harness supports but this paper does not report. The synthetic marks and attendance are random, so no educational conclusion can be drawn from demonstration outputs. The system is a single-user local application with no login or access control. Test-suite accuracy, which checks a query against many generated database instances, was not built; execution accuracy on one small database can overstate true semantic correctness when two different queries coincide on the same rows.

VII. CONCLUSION AND FUTURE WORK

This paper presented an AI-powered text-to-SQL chat application for education data analytics. Its design centres on five ideas drawn from the literature and from the risks of the education domain: LLM generation prompted with the schema and domain facts, a strict SELECT-only safety layer, executionguided self-correction, a deterministic rule-based fallback whose use is always labelled, and answers that show their SQL, explanation, and data together. On a synthetic university database the system achieves 10/10 execution accuracy on its own gold set, reported with its in-domain scope. Future work proceeds in four steps: enlarge the gold set to 60–100 questions stratified by difficulty, including deliberately ambiguous questions; add a retrieval-based schema linker and an education glossary, and measure each addition as an ablation; evaluate a live local code model against the baseline on the enlarged set, with error analysis by failure class; and evaluate the enlarged system on OULAD [13] and in a small usability study with non-technical users, including whether they can detect a deliberately wrong query. Those steps would convert a scoped demonstration into evidence about education NL2SQL in general—the benchmark contribution the field currently lacks.

ACKNOWLEDGMENT

The author thanks the faculty and peers who reviewed the project demonstration and the gold question set. No real student data was used at any stage of this work.

REFERENCES

[1] V. Zhong, C. Xiong, and R. Socher, “Seq2SQL: Generating structured queries from natural language using reinforcement learning,” arXiv:1709.00103, 2017.

[2] T. Yu et al., “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task,” in Proc. EMNLP, 2018, pp. 3911–3921.

[3] J. Li et al., “Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs,” in Proc. NeurIPS
(Datasets and Benchmarks), 2023. (BIRD.)

[4] F. Lei et al., “Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows,” in Proc. ICLR, 2025.

[5] B. Wang, R. Shin, X. V. Lin, O. Polozov, and M. Richardson, “RATSQL: Relation-aware schema encoding and linking for text-to-SQL parsers,” in Proc. ACL, 2020, pp. 7567–7578.

[6] H. Li, J. Zhang, C. Li, and H. Chen, “Decoupling the skeleton parsing and schema linking for text-toSQL,” in Proc. AAAI, 2023. (RESDSQL.) arXiv:2302.05965.

[7] M. Pourreza and D. Rafiei, “DIN-SQL: Decomposed in-context learning of text-to-SQL with selfcorrection,” in Proc. NeurIPS, 2023. arXiv:2304.11015.

[8] D. Gao et al., "Text-to-SQL empowered by large language models: A benchmark evaluation," Proc. VLDB Endowment, vol. 17, no. 5, pp.
1132–1145, 2024. (DAIL-SQL.)

[9] H. Li et al., "CodeS: Towards building open-source language models for text-to-SQL," Proc. ACM Manag. Data, vol. 2, no. 3, 2024. arXiv:2402.16347.

[10] Y. Gao et al., "XiYan-SQL: A multi-generator ensemble framework for text-to-SQL," arXiv:2411.08599, 2024.

[11] B. Wang et al., "MAC-SQL: A multi-agent collaborative framework for text-to-SQL," arXiv:2312.11242, 2023.

[12] S. Talaei et al., "CHESS: Contextual harnessing for efficient SQL synthesis," arXiv:2405.16755, 2024.

[13] J. Kuzilek, M. Hlosta, and Z. Zdrahal, "Open University Learning Analytics Dataset," Scientific Data, vol. 4, Art. 170171, 2017.

[14] X. Dong et al., "A survey on employing large language models for textto-SQL tasks," arXiv:2407.15186, 2024.

[15] B. Hui et al., "Next-generation database interfaces: A survey of LLMbased text-to-SQL," arXiv:2406.08426, 2024.

Copyright & License:

© Authors retain the copyright of this article. This work is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.