Lyra Computer Science and Artificial Intelligence

RESEARCH ARTICLE

OPEN ACCESS

LLM-Driven Query Planning for Intelligent Access to Structured and Semi-Structured Data

Mykola Khoroshevskyi [1,*]

[1] Sigma Software, Altea, Spain

* Correspondence: mykola.khoroshevskyi@icloud.com

Received: 17 August 2026 · Revised: 16 September 2026 · Accepted: 30 September 2026 · Published: 16 October 2026

https://doi.org/10.68660/lcsai.1.1.1

ABSTRACT

This paper addresses intelligent access to structured and semi-structured data using large language models (LLMs). We develop an approach to LLM-driven query planning in which a user's information need is handled as a sequence of interdependent operations: query analysis, grounding in data sources and the schema, plan construction, execution, validation of intermediate results, and, when needed, replanning. The study draws on a comparative analysis of the open benchmarks BIRD, Spider 2.0, and HybridQA and of representative published approaches, from Text-to-SQL pipelines such as DIN-SQL and CHASE-SQL to hybrid methods such as BlendSQL and Weaver. The proposed architecture separates planning from execution and allows SQL operations to be combined with semantic processing by an LLM. We formalize the plan as a directed acyclic graph of typed SQL, semantic, and lookup operations with validation predicates, define the conditions for partial replanning, and illustrate the procedure with a worked example. Our analysis of existing experimental results indicates that decomposing complex queries, combining heterogeneous sources, and monitoring intermediate state are each associated with higher accuracy or lower cost in multi-step data access tasks. The evidence supports assigning the LLM the role of a control component that coordinates the sequence of actions, although the proposed architecture still needs to be evaluated quantitatively under uniform experimental conditions. To that end, we propose an evaluation protocol with baselines, ablation variants, and falsifiable predictions.

KEYWORDS

large language models; LLM agents; query planning; Text-to-SQL; hybrid question answering; structured data; semi-structured data

1. Introduction

Large language models (LLMs) have broadened the possibilities for accessing data through natural-language queries. One of the most widely studied directions is Text-to-SQL, the automatic translation of a user's question into an SQL query. Solving such tasks correctly, however, takes more than generating syntactically valid SQL. The system also has to understand the database structure, map natural-language concepts onto schema elements, and interpret the values stored in the tables. How difficult this is can be seen

© 2026 The authors. Published by Lyra Science. CC BY 4.0.

from the progression of benchmarks such as Spider, BIRD, and Spider 2.0, which are built around multi-table and large-scale databases [1–3].

In real-world applications, single-step translation of natural language into SQL is no longer enough. Complex user requests may involve analyzing several tables, running a sequence of dependent operations, consulting additional metadata, and revising the initial query. Spider 2.0 covers enterprise-level tasks in which a single workflow may include several SQL queries, reading documentation, and successive data transformations [3]. This points to the need to move from simply generating a query to a controlled, multi-stage process of building and executing it.

A further difficulty arises when structured and semi-structured data must be used together. The required information may be spread across relational tables, text passages, and other loosely structured representations. The HybridQA benchmark, for example, contains questions that can be answered only by combining tabular data with the text linked to it [4]. An intelligent system therefore has to do more than formulate a query: it must also decide which sources and processing methods to use at each stage of the solution.

A widely explored remedy is to decompose a complex query into a sequence of simpler actions. DIN-SQL uses separate stages for schema linking, complexity classification, decomposition, and self-correction of the SQL query [5]. Studies of how LLMs interact with databases have also shown that planning a sequence of actions and generating several interrelated SQL queries remain serious problems for current LLM agents [6]. Hybrid approaches such as BlendSQL and Weaver further show that conventional operations on structured data can be combined with semantic processing by an LLM [7, 8].

This motivates the design of a mechanism in which the LLM not only generates the final query but also acts as a planner, determining the sequence of operations, the data sources required, and the dependencies between stages. In this paper, LLM-driven query planning means constructing an explicit multi-step plan that covers selecting relevant sources, decomposing the user's request, building executable operations, and revising them, if necessary, based on execution results.

The aim of this study is to develop, and to ground analytically in published evidence, an approach to LLM-driven query planning for intelligent access to structured and semi-structured data. We hypothesize that explicit multi-step planning that considers source metadata and intermediate execution results can make the processing of complex information requests more reliable than single-step generation of the final query. In operational terms, "more reliable" means higher accuracy on each benchmark's native metric, a smaller share of questions that exhaust the replanning budget, and fewer errors of the five types identified in Section 4.3, all measured under the protocol in Section 4.5.

2. Related Work

2.1. Decomposition and Multi-Agent Pipelines for Text-to-SQL

Early work on LLM-based Text-to-SQL concentrated on prompt design. Gao et al. [9] systematically compared ways of representing the question and the schema, selecting in-context examples, and organizing those examples in the prompt. They combined the most effective options in DAIL-SQL, which reached 86.6% execution accuracy on the Spider leaderboard with GPT-4 and self-consistency, and they paid explicit attention to the trade-off between accuracy and token cost [9]. In this line of work the query remains a single generation target: the model receives a carefully assembled prompt and returns one SQL statement.

DIN-SQL [5] was among the first methods to split this single step into stages, and later systems pushed the idea further by assigning stages to separate agents. MAC-SQL [10] uses three of them: a Selector that reduces a large schema to its relevant part, a Decomposer that breaks the question into sub-questions and builds the SQL step by step, and a Refiner that corrects the query using execution feedback. CHESS [11] follows a similar logic, with specialized agents for information retrieval, schema selection, candidate generation, and unit testing. CHASE-SQL [12] moves the emphasis to test-time computation. Several generators produce candidate queries using divide-and-conquer prompting, reasoning over query execution plans, and synthetic few-shot examples tailored to each instance, and a fine-tuned selection model chooses among them through pairwise comparison. CHASE-SQL reported 73.0% execution accuracy on the BIRD test set [12].

These systems share two features that are relevant for the present study. First, their decomposition is defined in advance by the pipeline designer: the set of stages is the same for every question, and the question can at most select among predefined branches. Second, they all aim at a single output, one SQL query, even when sub-queries are produced along the way. Neither assumption holds in the Spider 2.0 setting [3], where the number and kind of steps depend on the task and some steps involve reading documentation rather than querying data.

2.2. Neuro-Symbolic Methods and Table Reasoning

A second line of research asks what to do when SQL alone cannot express the question. Binder [13] maps the input to a program in SQL or Python whose grammar is extended with calls to a language model; the LLM thus acts both as the parser that writes the program and as a function invoked while the program runs. Its authors point out that, according to the SQUALL annotations, about 20% of WikiTableQuestions (WikiTQ) questions cannot be written in SQL at all, and Binder achieved state-of-the-art results on WikiTQ and TabFact with only a few dozen annotated examples and no task-specific training [13]. BlendSQL [7] develops the same idea into a dialect of SQLite in which LLM functions appear as ordinary query operators.

Chain-of-Table [14] treats the table itself as the carrier of intermediate reasoning. The model selects an operation, applies it to the table, and then plans the next operation on the transformed table, so that the sequence of evolving tables records how the answer was reached [14]. StructGPT [15] separates reading from reasoning: specialized interfaces extract relevant evidence from tables, knowledge graphs, or databases, and the LLM reasons over that evidence iteratively until it can return an answer or an executable SQL query. Weaver [8] goes a step further by making the plan explicit and allowing SQL and LLM operations to alternate within it.

Compared with Text-to-SQL pipelines, these methods are more flexible about the kinds of operations they allow, and several of them choose the next step dynamically. Their evaluation, however, is concentrated on table question answering. Binder and Chain-of-Table, for example, were tested mainly on single-table benchmarks such as WikiTQ and TabFact [13, 14], so the difficulties of schema linking in large multi-table databases remain largely outside their scope.

2.3. LLM Agents and Planning

ReAct [16] interleaves reasoning traces with actions. The model writes a thought, issues an action such as a search call, receives an observation, and uses it to revise its plan before acting again. This cycle of acting and observing is the closest general-purpose precedent for the Execute → Observe → Re-plan loop discussed in this paper. Nan et al. [6] examined how LLM

agents combine reasoning and action in database question answering and found that planning a sequence of actions and producing several related SQL queries remain difficult. Spider 2.0 [3] makes the same point at enterprise scale: agents that work in a full environment with documentation and several SQL dialects solve only a small share of the tasks.

In ReAct-style agents, though, the plan lives only in the text the model generates. There is no separate structure that can be inspected, checked, or partly re-executed. For data access this is a real limitation. An intermediate SQL result is a typed object with a schema and a row count, and it can be checked mechanically before the next step begins; a free-text thought cannot.

Some agent frameworks do make the plan explicit. LLMCompiler [17] has the LLM produce a directed acyclic graph of function calls, runs independent calls in parallel, and can replan on the basis of intermediate results; its authors report lower latency and cost and higher accuracy than ReAct. ReWOO [18] separates a Planner, which writes the full chain of interdependent tool calls in advance, from Workers that execute the calls and a Solver that composes the answer, which reduces token consumption. The present work shares this plan-then-execute structure but differs in three respects. Its operations are typed (SQL, SEM, LOOKUP), so relational work is delegated to the database engine rather than treated as a generic tool call. Each operation carries a validation predicate that is checked before its result is used downstream. Replanning is triggered by a failed check or by new knowledge about the data sources, and it regenerates only the part of the plan that depends on the failure.

2.4. Positioning of This Work

A note on terminology is needed first. In database systems, query planning usually refers to the work of the query optimizer, which chooses an efficient physical execution plan for a given declarative query [19]. In this paper, the term is used at a higher level: the plan determines which operations, over which sources, are needed to answer a natural-language request. Each SQL operation in such a plan is still optimized by the database engine in the usual way.

The literature reviewed above leaves three gaps. Text-to-SQL pipelines decompose queries, but through a set of stages defined in advance and toward a single SQL output. Neuro-symbolic and table reasoning methods mix structural and semantic operations, but they are rarely examined on large multi-table schemas. General-purpose agents adapt their actions to observations, but they either keep the plan implicit, as in ReAct, or make it explicit as a graph of untyped tool calls without validation conditions, as in LLMCompiler and ReWOO. The present work brings these strands together. Its contribution is architectural rather than a new model: (i) a formal model of the query plan as a graph of typed operations with explicit validation predicates (Section 3.3); (ii) rules for building and revising such plans, illustrated with a worked example (Sections 3.4 and 3.5); (iii) a qualitative comparison of representative systems against four design-level criteria (criteria defined in Section 3.7, comparison in Section 4.2); and (iv) a protocol for the controlled experiment needed to test the architecture quantitatively (Section 4.5).

3. Materials and Methods

3.1. Study Design and Source Materials

The study is designed as a comparative analytical assessment of LLM-based data access methods, followed by a formal description of an architecture for LLM-driven query planning. Its empirical basis consists of open benchmarks and the results reported in the original studies that introduced them. This design makes it possible to analyze the effects of schema complexity,

multistep execution, and source heterogeneity without mixing results obtained under different experimental conditions (see Section 3.6).

For structured data, we chose BIRD, a Text-to-SQL benchmark of 12,751 question–SQL pairs over 95 databases that total 33.4 GB and span more than 37 domains [2]. BIRD tests not only how well a natural-language question is mapped to the schema but also how a system handles actual database values, external knowledge, and the efficiency of SQL execution.

For enterprise-level tasks, we used Spider 2.0. It contains 632 tasks derived from real-world Text-to-SQL workflows. Its databases are hosted on various database management systems and cloud platforms, including BigQuery, Snowflake, SQLite, and DuckDB, and some schemas contain more than 1,000 columns. Solving these tasks may require processing metadata, reading documentation, and writing several interrelated SQL queries [3].

To evaluate access to heterogeneous information, we used HybridQA. In this dataset, each question is tied to both a Wikipedia table and text passages about the entities listed in that table. The questions are constructed so that answering them requires both the tabular and the textual information [4]. We therefore treat HybridQA not as a Text-to-SQL benchmark in the usual sense but as a means of studying tasks in which a structured representation is supplemented by less formalized text.

The selected materials thus cover three modes of increasing complexity: generating SQL over a large database, executing a multistep enterprise workflow, and answering questions from information presented in more than one form.

3.2. Architecture for LLM-Driven Planning

The proposed method is built on the sequence Analyze → Ground → Plan → Execute → Observe → Validate → Re-plan → Answer (Figure 1).

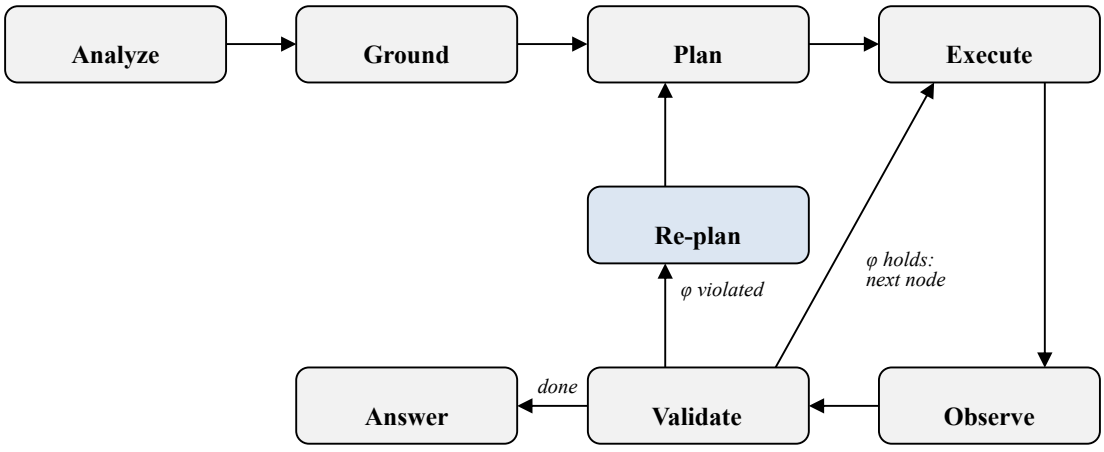


Figure 1. Architecture of LLM-driven query planning. When all nodes have been executed and validated, Validate passes control to Answer. After each validated step, control returns to Execute for the next ready node; a violated predicate φ sends control to Re-plan, which regenerates only the unexecuted part of the plan.

At the Analyze stage, the system identifies the user's information need: the entities involved, constraints, time conditions, aggregates, and the expected answer format. The Ground stage maps elements of the natural-language request to the available tables, columns, values, and other sources.

The Plan module then builds a sequence of operations and the dependencies between them. Actions that can be fully expressed in the relational model are carried out as SQL

operations: selection, filtering, joins, grouping, sorting, and aggregation. When text content has to be interpreted, the plan allows a semantic operation performed by the LLM. Recent hybrid approaches separate structural and semantic computation in a similar way: BlendSQL combines the processing of structured and unstructured data in a single interpretable query, and Weaver produces a flexible step-by-step plan that mixes SQL with LLM processing [7, 8].

The Execute stage runs the operation that has been formed. Observe records its actual result, and Validate then checks whether the operation succeeded and whether the result matches what the current step was expected to produce. If the operation fails, returns insufficient information, or reveals that additional data are needed, control passes to the Re-plan module, which revises the part of the plan that has not yet been executed.

The plan, then, is not fixed once it has been generated. It is refined as execution results come in. This is what fundamentally distinguishes the proposed model from a single-step scheme in which the language model produces the final SQL query in one pass.

3.3. Formal Model of the Plan

To make the architecture precise enough to implement and test, we describe the plan as a directed acyclic graph P = (V, E). Each node v ∈ V is an operation, and an edge (u, v) ∈ E means that v consumes the output of u. An operation is described by the tuple v = ⟨τ, s, in, out, φ⟩, where τ ∈ {SQL, SEM, LOOKUP} is the operation type; s is its specification (an SQL statement, a prompt for a semantic operation, or a request to a metadata or document source); in lists its inputs; out is the expected output type, which for SQL operations is a schema with column names and types; and φ is a validation predicate that the result must satisfy.

The three operation types follow the division of labor described in Section 3.2. SQL operations are executed by the database engine and cover everything the relational model can express. SEM operations are executed by the LLM and are reserved for interpreting text, normalizing values, and judging relevance. LOOKUP operations read metadata, documentation, or schema descriptions. They return no answer data, but they can change what the planner knows about the sources.

Execution proceeds over a state $S_t = \langle P_t, R_t, K_t, b_t, c_t \rangle$. Here P_t is the current plan, R_t maps executed nodes to their results, K_t is the accumulated knowledge about the sources (for example, the columns found relevant during grounding), and b_t is the number of replanning rounds still allowed. A separate limit B_c caps the total number of model calls per question, including those made by Analyze, Ground, Plan, and Re-plan, and c_t counts the calls made so far. A node is ready when all of its predecessors have results in R_t. Once a node has been executed, its result r is checked against φ. Useful predicates are usually simple and cheap to evaluate: the query ran without error, the result is not empty, the row count lies within an expected range, the output columns match out, or a key passed to a semantic step actually occurs in the referenced text.

Replanning is triggered by one of three events: an execution error, a violated predicate φ, or a LOOKUP or SEM result that adds to K_t information the current plan did not consider. In practice, new_knowledge(r) can be decided by simple checks: the result names a table, column, or value that is absent from the grounding G, refers to an entity that no node of the current plan uses, or contradicts an assumption recorded in K_t, such as a column type. A given piece of new information triggers replanning at most once, which limits the risk of the unstable replanning discussed in Section 4.3. When this happens, only the subgraph of nodes that have not yet been executed is regenerated; a node whose result failed validation is treated as unexecuted, so it is regenerated together with its descendants. Results already stored in R_t are kept and passed to

the planner as context, so completed work is not repeated. The budget b_t keeps the replanning loop from running indefinitely, and B_c bounds the overall cost; when either is exhausted, the system returns a failure report together with the partial results instead of an unsupported answer. Algorithm 1 summarizes the procedure.

Algorithm 1. LLM-driven query planning with partial replanning

Input: user request q, data sources D, budgets B (replanning) and B_c (calls)
Output: answer a, or a failure report

$I \leftarrow \text{Analyze}(q)$ ▷ entities, constraints, format
$G \leftarrow \text{Ground}(I, D)$ ▷ relevant schema and texts
$P \leftarrow \text{Plan}(I, G)$; $R \leftarrow \emptyset$; $K \leftarrow G$; $b \leftarrow B$
while P has unexecuted nodes **do**
  if model calls used $\geq B_c$ **then return** failure report with R ▷ call budget
  $v \leftarrow$ next ready node of P
  $r \leftarrow \text{Execute}(v, R)$ ▷ SQL engine, LLM, or lookup
  $R[v] \leftarrow r$ ▷ Observe
  if $\text{error}(r)$ **or not** $\varphi_v(r)$ **or** $\text{new_knowledge}(r, K)$ **then** ▷ Validate
    if $\text{error}(r)$ **or not** $\varphi_v(r)$ **then** $R \leftarrow R \setminus \{v\}$ ▷ failed node is regenerated
    $K \leftarrow K \cup \text{info}(r)$ ▷ update source knowledge
    if $b = 0$ **then return** failure report with R
    $P \leftarrow \text{Replan}(P, R, I, K)$; $b \leftarrow b - 1$ ▷ Re-plan
  end if
end while
return $\text{Answer}(I, R)$

3.4. Rules for Building and Revising the Plan

The plan is built in four steps. First, the system determines what data are needed for the answer and whether they are available in a single source. Next, it identifies the minimal set of tables, attributes, or linked text elements. The task is then decomposed into actions, each with a defined input and a verifiable result. Finally, the dependencies between actions are established.

Suppose, for example, that a user asks for the object with the highest value of some indicator and then for a property of that object described in linked text. First, the system runs an SQL aggregation and identifies the relevant record. Only the resulting identifier or name is passed to the next, semantic step. In this case there is no need to send the full contents of all sources to the LLM.

Explicit decomposition of this kind is methodologically similar to DIN-SQL, which splits a Text-to-SQL task into schema linking, complexity classification, generation, and SQL self-correction [5]. In the present work, however, decomposition extends beyond SQL generation: it also covers the choice among different types of operations and the possibility of restructuring the rest of the plan after an intermediate action has been executed.

3.5. Worked Example

To show how the model works on a concrete case, consider a question in the style of HybridQA: “Which stadium in the table has the largest capacity, and in what year did the club that plays there win its first national title?” The table lists stadiums with their clubs and capacities, and

each club name is linked to a short text passage about the club. The example was constructed for illustration and is not taken from the dataset.

Analyze identifies two sub-goals (an argmax over a numeric attribute and a fact stated in text) and an answer format consisting of a stadium name and a year. Ground maps "capacity" to the Capacity column, "club" to the Club column, and "first national title" to the passages linked from that column. Plan then produces the four-step graph shown in Table 1.

Table 1. Initial plan for the example question

Step	Type	Operation	Validation predicate φ
1	LOOKUP	Read the column descriptions; confirm that Capacity and Club exist and record their declared types	Both columns found
2	SQL	SELECT stadium, club, capacity FROM stadiums ORDER BY capacity DESC LIMIT 1	One row; the returned capacity parses as a number, and the type recorded in step 1 is numeric
3	SEM	In the passage linked to the club from step 2, find the year of the first national title	A four-digit year occurs in the passage
4	SEM	Compose the answer from the outputs of steps 2 and 3	Stadium name and year present

Note: constructed example; SQL is shown in simplified form. Because the answer format is fixed, step 4 could equally be a deterministic template; it is shown as SEM only for generality.

When step 2 runs, the validation predicate fails. Step 1 has recorded that Capacity is declared as text, and the returned value is the string "45,000" rather than a number. Because the column stores numbers with thousands separators, the values sort lexically, and the row returned is not the largest stadium. This is precisely the kind of error that a single-pass system would not notice, because the query is syntactically valid and returns a plausible-looking row. In the proposed model, the violated predicate triggers Re-plan. Only the unexecuted part of the graph is regenerated: a new SQL step casts Capacity to an integer after removing the separators, and step 2 is rewritten to sort on the converted value. Step 1 is not repeated, since its result is already in R_t.

Step 3 then shows the second benefit of the design. Only the club name selected in step 2 is passed to the semantic operation, together with the one linked passage, rather than the whole table and all passages. If the passage says "first league championship" instead of "first national title," the SEM operation can still recognize the equivalence; if it finds no year at all, the predicate fails, and the planner can add a LOOKUP step that searches the passages of related entities. In either case, every decision leaves a record in the plan and in R_t, which makes the final answer traceable to specific operations.

3.6. Comparative Analysis Method

Because the published studies rely on different language models, datasets, and experimental configurations, their absolute scores were not combined into a single aggregate metric. Numerical results are reported only with the values and units given in the corresponding primary publications.

For BIRD and other Text-to-SQL tasks, we examined Execution Accuracy (EX), the proportion of examples for which executing the generated SQL yields the correct result. For HybridQA, we considered Exact Match (EM) and F1, the metrics the benchmark's authors use to score answers. For Spider 2.0, we used the proportion of tasks solved in a specific published configuration.

Token consumption and the number of API calls were recorded separately and were not treated as measures of accuracy.

This procedure rules out invalid direct comparisons, such as setting one system's execution accuracy against another system's exact match. For this reason, the Results and Discussion section compares mainly the patterns observed: the effects of decomposition, of more complex workflows, of combining different sources, and of dynamic operation selection.

3.7. Analysis Criteria

Five criteria were defined to assess the applicability of LLM-driven planning:

1. Correctness of the final result: whether the sequence of actions leads to the required answer;
2. Correctness of schema linking: whether the necessary tables, fields, and values were selected;
3. Quality of decomposition: whether the complex request was split into logically complete operations;
4. Plan adaptability: whether the system can change the sequence after receiving intermediate results;
5. Efficiency of LLM use: whether the language model handles only those operations that actually require semantic interpretation.

These criteria are used below to interpret the published experimental results and to compare architectural approaches, not to produce an artificial overall ranking.

3.8. Reproducibility and Availability of Materials

The study uses only publicly documented benchmarks and the results of their primary publications. BIRD is distributed through the official benchmark project, Spider 2.0 through its official repository and project website, and HybridQA together with its publication and associated research materials [2–4]. No new personal, medical, or other participant data were collected, and no experiments involving humans or animals were carried out, so separate ethical approval was not required for this analysis.

Because the quantitative values in the Results and Discussion section are taken from published work, they are not presented as results of independently rerunning the models in question. For each value, the original benchmark, model, metric, and experimental context are preserved. This avoids attributing to the proposed architecture results that were in fact obtained by other authors.

4. Results and Discussion

4.1. Published Evidence

Published experimental results suggest that decomposing a complex query into a sequence of separate operations is an important way to improve Text-to-SQL systems. DIN-SQL splits the original task into schema linking, complexity classification, SQL generation, and self-correction. Its authors reported an execution accuracy of 85.3% on the Spider test set and 55.9% on BIRD. According to the study, decomposition improved execution accuracy over simple few-shot prompting by roughly 10% in relative terms for each of the three LLMs examined; on the Spider development set this corresponded to gains of 4.5 to 8.4 percentage points (for GPT-4, from 67.4% to 74.2%) [5]. These figures are taken from the DIN-SQL experiments and serve here only as evidence on the effectiveness of multi-stage query generation. Later pipelines that

split the task further report higher scores on BIRD: 59.6% execution accuracy on the test set for MAC-SQL and 73.0% for CHASE-SQL [10, 12].

Table 2 summarizes published results for query decomposition and planning approaches.

Table 2. Published results of query decomposition and planning approaches

Approach	Dataset	Metric	Result
DIN-SQL	Spider	Execution Accuracy	85.3%
DIN-SQL	BIRD	Execution Accuracy	55.9%
DAIL-SQL	Spider	Execution Accuracy	86.6%
MAC-SQL	BIRD	Execution Accuracy	59.6%
CHASE-SQL	BIRD	Execution Accuracy	73.0%

Source: compiled by the author from [5, 9, 10, 12]; test-set results. BlendSQL, which reports a cost measure rather than accuracy (35% fewer tokens than end-to-end baselines [7]), is discussed in the text.

Table 2 should not be read as a single ranking of the systems listed, since the results were obtained on different datasets with different models and experimental configurations. The drop from 85.3% on Spider to 55.9% on BIRD for the same method does, however, show how the complexity of the environment affects what LLMs can do. The Spider 2.0 experiment makes the effect even clearer: the o1-preview-based Code Agent described by its authors solved 91.2% of Spider 1.0 tasks, 73.0% of BIRD tasks, and only 21.3% of Spider 2.0 tasks (Figure 2) [3].

This drop reflects a fundamental increase in task complexity. Spider 2.0 contains 632 tasks based on enterprise-level scenarios. Its databases often have more than 1,000 columns, and individual tasks require working through metadata, documentation, and different SQL dialects, as well as writing several queries that transform data in succession [3]. The Spider 2.0 results thus expose the limits of an approach that treats the LLM merely as a generator of a single final SQL query.

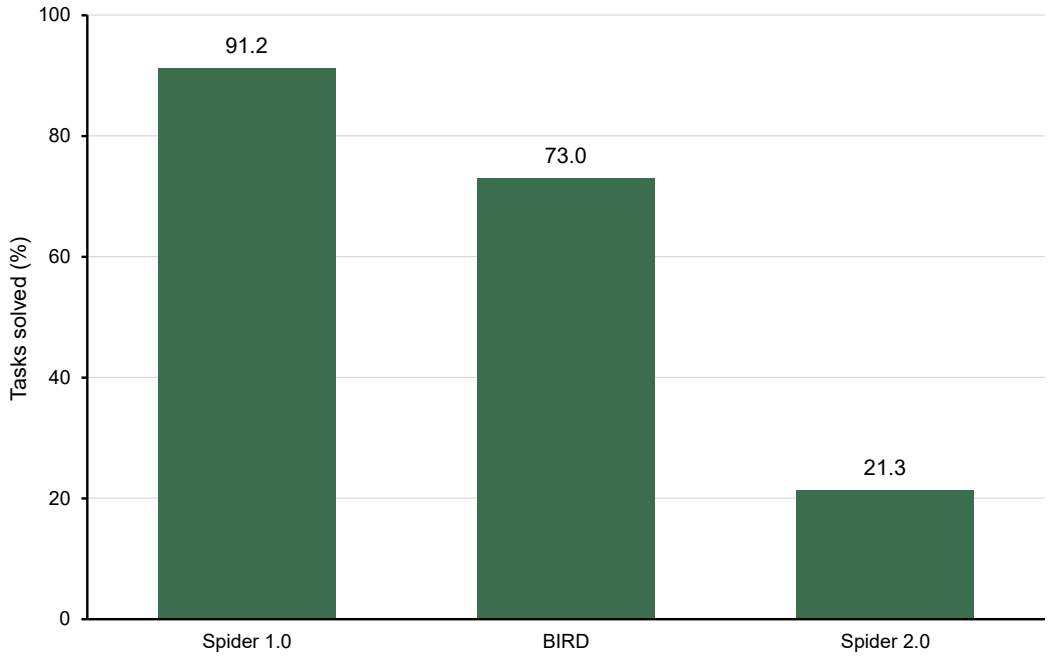


Figure 2. Share of tasks solved by the o1-preview-based Code Agent on Spider 1.0, BIRD, and Spider 2.0. Source: [3].

Figure 2 shows how sharply the performance of one and the same agent system falls when the task moves from the conventional Text-to-SQL setting to enterprise workflows. The decline should not be seen solely as a weakness in the model's ability to generate SQL. Spider 2.0 places additional demands on finding the relevant schema elements, working with external documentation, planning multiple queries, and transforming results step by step. The ability to construct a correct plan of action therefore becomes a performance factor in its own right.

A second important issue is access to information presented in different forms. The HybridQA results show that relying on a single type of data is not enough for tasks that require joint analysis of tables and the text associated with them. In the original HybridQA study, table-only and text-only baselines reached 8.3% and 19.1% Exact Match on the test set, whereas the hybrid model HYBRIDER, which used both sources, reached 43.8% [4].

Table 3 gives the HybridQA results for different types of information.

Table 3. HybridQA results for different types of information

Approach	Data used	EM, %	F1, %
Table-only baseline	Tables only	8.3	11.7
Text-only baseline	Text only	19.1	25.0
Hybrid model (HYBRIDER)	Tables and text	43.8	50.6

Source: compiled by the author from [4]; test set, HYBRIDER with BERT-large.

The results in Table 3 confirm that tasks in which information is split between structured and textual sources call for more than one processing method. The HybridQA authors deliberately designed the questions so that the correct answer could not be obtained if either type of information was missing. At the same time, the hybrid model's score of 43.8% EM shows that merging sources does not by itself solve the problem of constructing a correct chain of reasoning.

A natural way to organize this process is to separate structural and semantic operations. BlendSQL, an extension of SQLite, coordinates the processing of structured and unstructured data. The full decomposed chain of reasoning is expressed as an interpretable BlendSQL query. Its authors showed that the approach can improve on end-to-end systems while using 35% fewer tokens [7].

This result is also relevant to the LLM-driven planning considered here. Multi-step processing does not have to mean passing the full contents of every available source to the language model at each stage. If the planner first identifies the relevant tables, columns, or text passages and then hands only the necessary data to the next component, the context can be kept small. Planning then serves not only a logical function but also controls how much information is processed.

Weaver applies the idea of dynamic planning more directly. It builds a flexible step-by-step plan that uses SQL to retrieve structured data and an LLM for operations that require semantic interpretation. According to the published results, Weaver outperformed the methods it was compared with on four Table QA datasets while also reducing the number of API calls and the error rate. It needed about six API calls per question, against roughly eight for H-STAR and about fifty for Binder with self-consistency, and on WikiTQ it reduced SQL errors by 30% and plan generation errors by 86% relative to ProTrix [8].

Taken together, these results point to three principles that are significant for LLM-driven data access. The first is to decompose a complex user request into simpler, verifiable stages. The DIN-SQL results show that organizing the process this way can improve Text-to-SQL quality compared with generating the query directly through few-shot prompting. The second is to

divide operations by processing method: relational transformations are best handled in SQL, while interpreting unstructured or loosely structured content may require an LLM. The third concerns plan adaptability: the result of each intermediate step should inform the choice of subsequent actions.

4.2. Qualitative Comparison of Architectures

The numbers reported above cannot be pooled, but the systems behind them can be compared by design. Table 4 assesses representative approaches against the four architectural criteria from Section 3.7; the first criterion, correctness of the final result, is an outcome rather than a design property and is covered by the published metrics. The assessment reflects the author's reading of the original descriptions, not a measurement.

Table 4. Architectural properties of representative approaches

Approach	Schema linking	Decomposition	Plan adaptability	Selective LLM use
DIN-SQL [5]	+	+ (fixed stages)	± (final SQL only)	–
MAC-SQL [10]	+	+ (fixed agents)	± (execution-based refinement)	–
CHASE-SQL [12]	±	+	± (query fixing, candidate selection)	–
Binder [13], BlendSQL [7]	–	±	–	+
Chain-of-Table [14]	–	+	+	±
Weaver [8]	±	+	+	+
ReAct-style agents [3, 16]	±	±	+	–
Proposed architecture	+	+	+ (partial replanning)	+

Note: + explicit mechanism; ± partial mechanism, with its scope given in parentheses where relevant; – not addressed. Entries for the proposed architecture reflect design intent, not measured results.

Two patterns stand out in Table 4. Text-to-SQL pipelines score well on schema linking and decomposition, but their adaptability is limited to repairing the final query, and the LLM performs every stage, including those a database engine could do more cheaply and more reliably. Table reasoning methods have the opposite profile: they adapt the next step to intermediate results and restrict the LLM to semantic sub-tasks, but they are designed for single tables and largely skip schema linking. Weaver comes closest to combining both profiles, which agrees with its reported reduction in API calls [8]. The proposed architecture is meant to cover all four properties at once; whether it does so in practice is a question for the experiment outlined in Section 4.5.

The "+" for selective LLM use in the proposed architecture needs a qualification. Analyze, Ground, Plan, and every Re-plan round are themselves LLM calls, so the architecture pays a fixed planning overhead of at least three calls per question, plus one for each replanning round, before any data are processed. For a simple question that a single SQL query answers correctly, this overhead is pure cost, and single-step generation will be cheaper. The savings appear only when delegating relational work to SQL keeps large tables and passages out of the model's context, or when validation prevents a wrong answer that would otherwise require a full retry. Where the balance lies is an empirical question that the cost measures in Section 4.5 are designed to answer; a practical system could also route questions it classifies as simple directly to single-step generation.

4.3. Limitations and Failure Modes

Each additional step in a multi-stage pipeline is another potential point of failure and adds computational cost. The resulting risks can be grouped into five failure modes that an implementation of the proposed architecture would have to handle. The first is grounding error: a term in the request is linked to the wrong column or value, and every later step inherits the mistake. Validation predicates catch this only partly, because a wrong column often yields a well-formed, non-empty result. The second is structural error in the plan itself, such as a missing dependency, a redundant step, or an aggregation applied before a needed filter. The third is misassignment of operations: a semantic step used where SQL would suffice wastes tokens and introduces nondeterminism, while an SQL step used for a judgment that requires reading text produces confident but wrong results.

The fourth failure mode concerns validation. Predicates that are too weak let errors through, as with a lexically sorted numeric column that still returns one row; predicates that are too strict trigger needless replanning. The fifth is unstable replanning, in which the planner alternates between two flawed versions of the plan or keeps adding steps without converging. The budget b_t in Section 3.3 bounds the damage, but it does not prevent the behavior. Practical mitigations include sampling values from candidate columns during grounding, keeping a log of rejected plan fragments so that the planner does not regenerate them, and preferring SQL whenever an operation can be expressed relationally.

4.4. Threats to Validity

Several factors limit how far the conclusions of this study can be generalized. Construct validity is affected by the metrics: execution accuracy, exact match, and task success rate measure different things, and execution accuracy can reward a query that returns the right result for the wrong reason. Internal validity is limited because the cited results were obtained with different LLMs, prompts, and model versions, and because public benchmarks may have entered the training data of the models evaluated on them, which can inflate scores. External validity is limited by the benchmarks themselves. Wikipedia tables and curated databases are cleaner and better documented than most enterprise data, so the gap observed on Spider 2.0 may understate the difficulty of real deployments. Finally, the qualitative comparison in Table 4 reflects the author's reading of published descriptions and would benefit from independent assessment.

4.5. Proposed Evaluation Protocol

Testing the architecture quantitatively requires a controlled experiment in which only the planning strategy varies. We propose the following protocol. The evaluation would use the BIRD development set for large-schema Text-to-SQL, the Spider 2.0-Lite setting (547 text-to-SQL tasks over BigQuery, Snowflake, and SQLite databases) for multistep enterprise workflows, and the HybridQA development set for combined table and text reasoning [2–4]. All systems would run on the same LLM, with temperature set to zero, identical schema serialization, and the same limit on model calls per question.

Three baselines would be compared with the full architecture: single-step generation of the final query; a fixed decomposition pipeline in the style of DIN-SQL and MAC-SQL [5, 10]; and a ReAct-style agent with free-text planning [16]. Ablated variants, listed in Table 5, would isolate the contribution of each component.

Table 5. Planned ablation variants

Variant	Change relative to the full architecture	Question addressed
Full	—	Reference configuration
No Ground	Planner receives the full schema without grounding	How much does explicit grounding contribute?
No Validate	Predicates φ disabled; results are accepted as returned	Do validation checks catch errors that matter?
No Re-plan	Validation failures end the run instead of triggering replanning	What is the value of partial replanning?
SQL only	SEM operations removed	When is semantic processing necessary?
SEM only	SQL operations replaced by LLM processing	What does delegating relational work to SQL save?

Accuracy would be reported with the metric native to each benchmark: execution accuracy for BIRD, task success for Spider 2.0, and exact match and F1 for HybridQA. Cost would be measured by the number of model calls, input and output tokens, and wall-clock time per question, and planning behavior by the number of replanning rounds and the share of questions that exhaust the budget. Differences between configurations would be tested on paired per-question outcomes, using McNemar's test for binary correctness and bootstrap confidence intervals for continuous measures. A sample of failures from each configuration would also be annotated with the five failure modes from Section 4.3, so that the experiment shows not only whether the architecture helps but where it breaks.

The protocol makes three predictions that the experiment could refute. If explicit planning matters, the full architecture should outperform single-step generation most clearly on Spider 2.0 and HybridQA, where tasks need several dependent steps. If validation and partial replanning matter, the No Validate and No Re-plan variants should lose accuracy on those same benchmarks while saving relatively little cost. If the division of labor matters, the SEM only variant should consume substantially more tokens than the full architecture without a matching gain in accuracy.

5. Conclusions

The analysis shows that LLM-based access to complex data sources should be viewed not only as a matter of generating a final SQL query but also as a matter of managing a sequence of interrelated operations. As schemas grow, as processing comes to involve several dependent stages, and as tabular and textual information must be used together, four things become central: decomposing the user's request, choosing the relevant source, selecting an appropriate processing method, and checking intermediate results.

On this basis, we formulated an architecture for LLM-driven planning in which an answer is built through the sequence Analyze → Ground → Plan → Execute → Observe → Validate → Re-plan → Answer. Its defining feature is the separation of planning from execution: the language model determines which actions are needed and how they depend on one another, while the operations themselves are carried out by specialized tools, including SQL and semantic processing. Intermediate results can be used to refine later steps, so the plan is a dynamic structure rather than one fixed in advance.

A comparison of the published results for DIN-SQL, MAC-SQL, CHASE-SQL, BlendSQL, and Weaver on BIRD, Spider 2.0, and HybridQA supports several architectural principles: explicit

decomposition of complex queries, separate use of structural and semantic operations, and reliance on intermediate state when deciding on further actions. Existing studies also show, however, that more complex schemas and workflows still pose serious difficulties even for current LLM systems. A powerful language model alone, therefore, does not remove the need for controlled planning of data access.

A limitation of this study is that its quantitative analysis relies on results published by the authors of the respective benchmarks and methods. Because these results were obtained with different models, datasets, and experimental configurations, they cannot prove quantitatively that the proposed architecture, as a standalone implemented system, is superior. In this paper they serve to justify architectural decisions and to identify consistent patterns, not as the results of a unified experiment of our own.

To make such experiments possible, the paper goes beyond a verbal description of the architecture. It defines the plan as a graph of typed operations with explicit validation predicates, specifies when and how the unexecuted part of the plan is regenerated, and proposes an evaluation protocol with baselines, ablations, and predictions that the results could contradict. These elements turn the proposed approach into a set of testable claims rather than a general recommendation.

Future work should include a controlled experimental implementation of the proposed approach, with a fixed LLM and identical conditions for single-step and multi-step processing. Of particular interest is a separate assessment of the contributions of schema linking, decomposition, intermediate-result validation, and the Re-plan mechanism, along with an analysis of computational cost in terms of model calls, token usage, and execution time. Another direction is to extend the planner to other types of semi-structured sources and to develop confidence estimation mechanisms that can tell when a result is sufficient to stop and when the plan needs to be revised.

In sum, LLM-driven planning is an approach to natural-language data access in which the language model serves primarily to coordinate and choose actions rather than only to generate the final query. Its practical effectiveness remains to be confirmed by further reproducible experiments under uniform conditions.

Funding. This research received no external funding.

Data Availability Statement. No new data were created in this study. The benchmarks analyzed are publicly available from their original publications and project repositories [2–4], and the results discussed are taken from the cited primary studies [3–5, 7–10, 12].

Acknowledgments. During the preparation of this manuscript, the author used generative AI tools for language-related assistance, including structuring and editing the English text. The author reviewed and revised all output and takes full responsibility for the content of this publication.

Conflicts of Interest. The author declares no conflicts of interest.

References

1. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; Radev, D. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 31 October – 4 November 2018; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 3911–3921. https://doi.org/10.18653/v1/D18-1425

2. Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; Zhou, X.; Ma, C.; Li, G.; Chang, K. C.-C.; Huang, F.; Cheng, R.; Li, Y. Can LLM Already Serve as a Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 42330–42357. https://doi.org/10.52202/075280-1835
3. Lei, F.; Chen, J.; Ye, Y.; Cao, R.; Shin, D.; Su, H.; Suo, Z.; Gao, H.; Hu, W.; Yin, P.; Zhong, V.; Xiong, C.; Sun, R.; Liu, Q.; Wang, S. I.; Yu, T. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR 2025)*, Singapore, 24–28 April 2025. Available online: https://openreview.net/forum?id=XmProj9cPs (accessed on 12 June 2026).
4. Chen, W.; Zha, H.; Chen, Z.; Xiong, W.; Wang, H.; Wang, W. Y. HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online, November 2020; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 1026–1036. https://doi.org/10.18653/v1/2020.findings-emnlp.91
5. Pourreza, M.; Rafiei, D. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 36339–36348. https://doi.org/10.52202/075280-1577
6. Nan, L.; Zhang, E.; Zou, W.; Zhao, Y.; Zhou, W.; Cohan, A. On Evaluating the Integration of Reasoning and Action in LLM Agents with Database Question Answering. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Mexico City, Mexico, 16–21 June 2024; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 4556–4579. https://doi.org/10.18653/v1/2024.findings-naacl.284
7. Glenn, P.; Dakle, P.; Wang, L.; Raghavan, P. BlendSQL: A Scalable Dialect for Unifying Hybrid Question Answering in Relational Algebra. In *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand, August 2024; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 453–466. https://doi.org/10.18653/v1/2024.findings-acl.25
8. Khoja, R.; Gupta, D.; Fu, Y.; Roth, D.; Gupta, V. Weaver: Interweaving SQL and LLM for Table Reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, Suzhou, China, November 2025; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 28282–28308. https://doi.org/10.18653/v1/2025.emnlp-main.1436
9. Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; Zhou, J. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* **2024**, *17*, 1132–1145. https://doi.org/10.14778/3641204.3641221
10. Wang, B.; Ren, C.; Yang, J.; Liang, X.; Bai, J.; Chai, L.; Yan, Z.; Zhang, Q.-W.; Yin, D.; Sun, X.; Li, Z. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, Abu Dhabi, United Arab Emirates, 19–24 January 2025; Association for Computational Linguistics: Stroudsburg, PA, USA, 2025; pp. 540–557. Available online: https://aclanthology.org/2025.coling-main.36/ (accessed on 18 June 2026).
11. Talaei, S.; Pourreza, M.; Chang, Y.-C.; Mirhoseini, A.; Saberi, A. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *arXiv* 2024, arXiv:2405.16755. https://doi.org/10.48550/arXiv.2405.16755
12. Pourreza, M.; Li, H.; Sun, R.; Chung, Y.; Talaei, S.; Kakkar, G. T.; Gan, Y.; Saberi, A.; Ozcan, F.; Arik, S. O. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR 2025)*, Singapore, 24–28 April 2025. Available online: https://openreview.net/forum?id=CvGqMD5OtX (accessed on 23 June 2026).
13. Cheng, Z.; Xie, T.; Shi, P.; Li, C.; Nadkarni, R.; Hu, Y.; Xiong, C.; Radev, D.; Ostendorf, M.; Zettlemoyer, L.; Smith, N. A.; Yu, T. Binding Language Models in Symbolic Languages. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, Kigali, Rwanda, 1–5 May 2023. Available online: https://openreview.net/forum?id=lH1PV42cbF (accessed on 30 June 2026).
14. Wang, Z.; Zhang, H.; Li, C.-L.; Eisenschlos, J. M.; Perot, V.; Wang, Z.; Miculicich, L.; Fujii, Y.; Shang, J.; Lee, C.-Y.; Pfister, T. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*, Vienna, Austria, 7–11 May 2024. Available online: https://openreview.net/forum?id=4L0xnS4GQM (accessed on 7 July 2026).
15. Jiang, J.; Zhou, K.; Dong, Z.; Ye, K.; Zhao, W. X.; Wen, J.-R. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, 6–10 December 2023; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 9237–9251. https://doi.org/10.18653/v1/2023.emnlp-main.574

16. Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, Kigali, Rwanda, 1–5 May 2023. Available online: https://openreview.net/forum?id=WE_vluYUL-X (accessed on 14 July 2026).
17. Kim, S.; Moon, S.; Tabrizi, R.; Lee, N.; Mahoney, M. W.; Keutzer, K.; Gholami, A. An LLM Compiler for Parallel Function Calling. In *Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria, 21–27 July 2024; Proceedings of Machine Learning Research, Vol. 235; PMLR, 2024; pp. 24370–24391. Available online: https://proceedings.mlr.press/v235/kim24y.html (accessed on 21 July 2026).
18. Xu, B.; Peng, Z.; Lei, B.; Mukherjee, S.; Liu, Y.; Xu, D. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. *arXiv* 2023, arXiv:2305.18323. https://doi.org/10.48550/arXiv.2305.18323
19. Selinger, P. G.; Astrahan, M. M.; Chamberlin, D. D.; Lorie, R. A.; Price, T. G. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Boston, MA, USA, 30 May – 1 June 1979; Association for Computing Machinery: New York, NY, USA, 1979; pp. 23–34. https://doi.org/10.1145/582095.582099

Disclaimer. The views and conclusions in a published article are those of the authors and do not necessarily reflect the position of Lyra Science or its editorial board, neither of which accepts liability for any harm arising from methods, products, or instructions described in published content.