SANA: What Matters for QA Agents over Massive Data Lakes?

Austin Senna Wijaya, Jiaxiang Liu, Haonan Wang, Eugene Wu
DAPLab Columbia University
New York, USA
{asw2215,jl6235,hw2983,ew2493}@columbia.edu

ABSTRACT

Exploratory question answering (EQA) over data lakes requires an LLM agent to discover relevant sources, analyze retrieved data, and adapt its actions based on intermediate results. End-to-end accuracy alone cannot distinguish failures in search, planning, data analysis, or the agent's **Action Policy**: its decisions about what to do next and when to submit an answer. We present SANA (Search Agent Navigation Ablation framework), a diagnostic ablation framework that transforms EQA tasks into runtime profiles containing gold source sequence, sanitized subquestions, and execution records. SANA uses these profiles to construct idealized search, planning, and data-analysis tools, allowing each component to be ablated; the residual gap is diagnostic evidence for policy failures.

To illustrate SANA as a reusable evaluation framework, we adapted two recent EQA benchmarks, LakeQA and KramaBench, and evaluated lightweight and mid-sized agents under fixed prompts, budgets, data lakes, and runtimes. Across both benchmarks, data analysis is a consistent bottleneck while planning is less so. Search is a major limitation in LakeQA's large data-lake setting, but less so for the smaller-scale KramaBench. SANA thus deconstructs end-to-end task accuracies into a diagnosis of where data-lake agents fail, and allows for systematic comparisons of progress in search, planning, data analysis, and agent design.

VLDB Workshop Reference Format:
Austin Senna Wijaya, Jiaxiang Liu, Haonan Wang, Eugene Wu. SANA: What Matters for QA Agents over Massive Data Lakes?. VLDB 2026 Workshop: DASHSys: Systems for Data-centric Agents with Human-in-the-loop.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sana-ablation/sana-framework.

1 INTRODUCTION

Exploratory question answering (EQA) was recently introduced in LakeQA [20] as the problem where an agent must answer questions over facts that are distributed across a large data lake. The agent must iteratively reason about missing evidence that must be searched for, inspect and analyze retrieved data, and use intermediate findings to guide subsequent actions. This models practical analytics over data lakes: data scientists and business analysts can formulate high-level questions, but not the tables, filters, or computations needed to answer them.

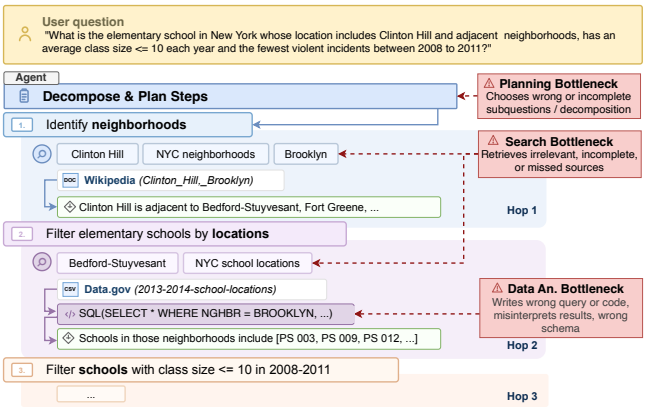

Figure 1: EQA agent runtime loop, with inherent bottlenecks during planning, searching, and data analysis.

Evaluating EQA is difficult because success depends not only on the quality of individual capabilities (e.g., search, planning, analysis of retrieved datasets), but also on how the agent composes them over a long trajectory. We refer to the agent's **Action Policy**, or simply **Policy**, as the turn-level decision process that selects the next action given the question, discovered sources, intermediate observations, tool errors, and remaining budget. At each turn, the policy must decide whether to search, inspect, compute, validate, revise, or submit an answer.

Prior work has advanced individual capabilities needed for EQA, including agentic retrieval [2, 10, 24], planning and decomposition [11, 14], tool use [15, 16], and code/SQL generation [8, 9, 23]. EQA requires these capabilities to interact in a single long-horizon loop [20]. Recent evaluations study facets of EQA: DCI [10] gives agents direct grep-style corpus access, while Metadata Reasoner [25] evaluates sufficient and minimal dataset selection. Neither isolates the contributions of search, planning, analysis execution, and agent policy to end-to-end accuracy. Consequently, failures from missing data, ignored evidence, incorrect decomposition, and brittle SQL or code remain conflated. SANA provides controlled ablations that diagnose these bottlenecks in end-to-end EQA agents.

To bridge this gap, we propose SANA (Search Agent Navigation Ablation framework), an ablation framework to study EQA agent policies. Specifically, we develop an idealized EQA agent whose trajectory planning, search system, and data analysis code are designed using the ground truth. This produces an optimistic upper bound, unattainable in practice, that isolates the agent's policy decisions, such as invoking tools and submitting its final answer.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

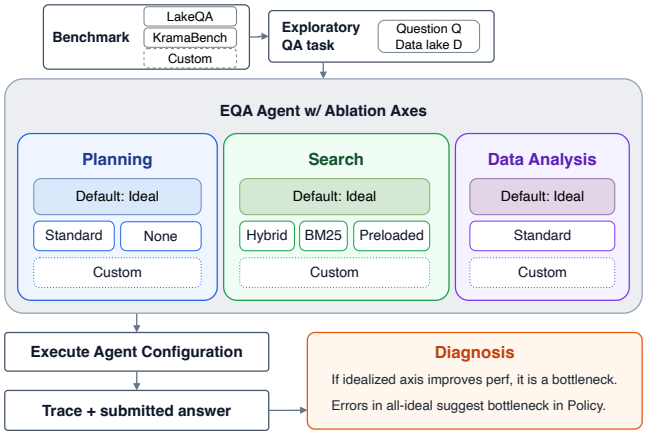

Figure 2: SANA's three runtime components as an ablation study. Each component can be evaluated with idealized, standard, baseline, or custom implementations.

We then systematically ablate each component with widely used implementations (e.g., BM25 or hybrid for search).

By idealizing individual components, SANA distinguishes failures caused by planning, search, or data-analysis execution from failures that remain in the agent policy. These residual policy failures include pursuing the wrong source, failing to validate intermediate evidence, submitting an incorrect final answer despite useful evidence, or exhausting the budget without progress. They expose concrete targets for future EQA agents, including stronger evidence tracking, validation, final-answer checking, and stopping criteria.

To summarize, we make the following contributions:

(1) **SANA**, a diagnostic ablation framework that converts solved EQA tasks into runtime profiles and uses them to construct idealized planning, search, and data-analysis interfaces while preserving the agent policy.
(2) A **controlled evaluation** on LakeQA [20] and a conversion of KramaBench [7] that isolates the end-to-end contribution of planning, search, and data-analysis execution under fixed agents, prompts, budgets, data lakes, and runtimes.
(3) An **empirical diagnosis of EQA bottlenecks**: data-analysis execution is a consistent limitation across both benchmarks; search becomes a major limitation in LakeQA's larger-scale discovery setting; and residual agent-policy failures remain even after component idealization.

2 RELATED WORK

Benchmarks for exploratory QA. QA benchmarks test multi-hop reasoning over heterogeneous data types, including HotpotQA [21], MuSiQue [18], OTT-QA [4], TAT-QA [26], and FeTaQA [12]. These benchmarks stress reasoning over documents, tables, or mixed evidence, but typically operate over a fixed corpus or compact evidence pool. Data-lake benchmarks move closer to exploratory analysis: LakeQA [20] requires agents to search, inspect, compute, and synthesize over large data lakes, while KramaBench [7] evaluates data-to-insight pipelines over multiple structured and unstructured sources. Recent agentic data-discovery evaluations such as DCI [10] and Metadata Reasoner [25] also study agents that interact with a corpus or select task-relevant datasets, but they do not isolate search, planning and execution as separate failure modes. SANA builds on this benchmark direction, but uses these tasks to diagnose which part of an EQA agent fails.

Component methods for data-lake QA. Prior work improves individual components needed for EQA. Dataset discovery systems such as D3L [3] and Starmie [6] retrieve joinable, unionable, or semantically related tables, while Pneuma [2] and AutoDDG [24] use LLM-generated descriptions to improve tabular retrieval. Text-to-SQL and data-agent benchmarks such as BIRD [9] and Spider 2.0 [8] evaluate data analysis execution once the relevant database context is known. Decomposition methods such as DecompRC [11] and DIN-SQL [14] expose intermediate structure for multi-hop reasoning or SQL generation. SANA treats these as interacting bottlenecks rather than isolated tasks: decomposition must be grounded and guide the agent's next discovery and analysis, discovery must feed analysis, and analysis must feed later discovery.

Agentic RAG and navigation. Agentic RAG systems use an LLM-controlled policy to decide when to retrieve, call tools, update context, and stop [17], building on reasoning-and-acting methods such as ReAct [22], IRCoT [19], and Self-RAG [1]. Recent systems such as MA-RAG [13] and A-RAG [5] further explore multi-agent or hierarchical retrieval architectures. These systems expose the same navigation decisions that arise in EQA, but their evaluations often conflate model policy, retrieval infrastructure, data profiling, tool design, and execution errors. SANA provides a controlled evaluation substrate by holding the task and runtime fixed while ablating search, planning, and execution.

3 THE SANA ABLATION FRAMEWORK

SANA is an ablation framework for search, planning, and computation—three major components in EQA. This section introduces the EQA task, motivates these components, and presents SANA's design.

3.1 Anatomy of an Exploratory QA Task

Here, we describe the structure of an EQA task and inter-dependencies between data discovery, question decomposition, data analysis, and tool implementation that are needed to successfully answer a task. In short, data discovery must retrieve the necessary datasets, but if it retrieves too many, then it makes question decomposition challenging. Similarly, if the data analysis tools are poorly implemented or not sufficiently expressive, then the agent cannot extract the necessary information from the datasets to determine answers or next steps.

3.1.1 An EQA Task. In LakeQA [20], an EQA task is defined as $\tau = (Q, \mathcal{D}, \mathcal{D}_{\text{gold}}, \mathcal{T}, A^\star, B)$, where Q is a natural-language question, $\mathcal{D} = \{d_1, \ldots, d_N\}$ is a large data lake containing structured and unstructured datasets, $\mathcal{T}$ is the set of tools available to an LLM agent, $A^\star$ is the gold answer to Q, $\mathcal{D}_{\text{gold}}$ contains a minimal set of datasets[1] sufficient to derive the gold answer, and the budget B is the maximum number of tool calls permitted.

[1] In practice, $\mathcal{D}_{\text{gold}}$ may not be unique, as it's currently intractable to enumerate all minimal sets. SANA inherits this $\mathcal{D}_{\text{gold}}$ from its underlying benchmarks (see Limitation (iv)).

Generating an answer $\hat{A}$ for an EQA task requires decomposing Q into a sequence of subquestions $(Q_1, Q_2, \ldots, Q_K)$, discovering each subquestion's gold dataset $(d^1_{\text{gold}}, \ldots, d^K_{\text{gold}})$, and answering $(A_1, A_2, \ldots, A_K)$, such that each intermediate answer can be derived from the current subquestion and gold dataset, and the previously derived answers:

$$Q_i, d^i_{\text{gold}}, A_1, \ldots, A_{i-1} \vDash A_i, \quad i = 1, \ldots, K. \tag{1}$$

Operationally, an agent must use the tools in $\mathcal{T}$ to discover candidate datasets, analyze them to assess their task-relevance, extract information to compute intermediate answers, and derive the final answer $\hat{A}$—all within the tool call budget. We next describe the classes of tools in $\mathcal{T}$ and the main challenges posed by EQA. Here, t denotes the t^{th} tool call made by the agent.

3.1.2 Agent tools. The tool set $\mathcal{T}$ supports two classes of operations: (i) dataset search $f(q, k)$, over the data lake and (ii) data analysis $g(c, d)$ over a dataset $d \in \mathcal{D}$. The search tool $f(q, k)$ currently is based on keyword search over dataset contents; it takes as input keywords q, and returns the top-k documents from the data lake. Analysis $g(c, d)$ takes as input an executable program (e.g., Python program, SQL query) c to evaluate over dataset d_i, and returns the corresponding execution I/O traces.

We use $f_t = f_t(q_t, k_t)$ and $g_t = g_t(c_t, d_t)$ as shorthand to denote the specific tool and its arguments in the t^{th} tool call. We denote the cumulative set of discovered datasets at turn t as $\mathbf{D}^t_{\text{disc}}$. In practice, an agent can analyze multiple datasets together, but the text refers to a single dataset for legibility.

3.1.3 Planning. The agent must decompose the EQA question by iteratively identifying the next subquestion to answer given the previous evidence. Formally, given the current discovered datasets $\mathbf{D}^t_{\text{disc}}$ at tool call t, planning produces an ordered sequence of subquestions and evidence choices

$$\big((\widehat{Q}_1, \widehat{d}_1), (\widehat{Q}_2, \widehat{d}_2), \ldots\big)$$

where $\widehat{d}_i$ may be unknown at the time $\widehat{Q}_i$ is first formulated and must be discovered through subsequent search calls.

3.1.4 Search. The EQA tasks are designed such that the gold evidence $\mathcal{D}_{\text{gold}}$ is necessary in order to correctly answer the subquestions and the task Q. Thus, the agent must, at a minimum, within a budget of B tool calls, identify a superset $\mathcal{D}^B_{\text{disc}} \supseteq \mathcal{D}_{\text{gold}}$. However, high recall alone is not sufficient: $\mathcal{D}^B_{\text{disc}}$ could degenerate to the entire data lake. Search precision is therefore also critical, because irrelevant datasets increase the burden on downstream planning and analysis.

3.1.5 Data analysis. Even if the agent retrieves the gold dataset for the current subquestion, it still needs to analyze that dataset to derive the intermediate answer. In short, the agent must correctly implement the entailment in eq. (1) to derive $\widehat{A}_i = g_t(\widehat{c}_i, \widehat{d}_i)$ without execution failure. If $\widehat{A}_i = A_i$ then the agent has correctly answered subquestion i.

3.1.6 Residual Failures. The three challenges above define the major failure classes targeted by SANA's ablations. However, they do not exhaust all ways an EQA task can fail. For example, even if the agent correctly derived the answers to the correct query decomposition, it may still hallucinate an incorrect final answer $\widehat{A} \neq A^\star$. They may also waste the budget by completing irrelevant actions, skipping necessary steps, or hallucinating. We discuss these residual failures in Section 5.3.

3.2 SANA Ablation

SANA isolates the effect of the major components in EQA (section 3.1) by ablating search, planning, and data analysis in isolation or combination. Ablation replaces a component with an idealized version or other implementations. This section first motivates *intent-based* ablation, then the task profile used to generate idealized components, and finally the ablation method and implementation for each.

3.2.1 Intent-based Ablation. Isolating the contribution of the system components alone requires careful design. For instance, a naive approach to ablate search is to simply give $\mathcal{D}_{\text{gold}}$ to the agent. However, the agent should still be expected to formulate the appropriate search query; we aim to control only the search tool's quality.

SANA therefore extracts the semantic intention a_t as natural language behind each tool call. For search, SANA extracts a_t as the keyword q; for execution, SANA extracts a_t as the analysis goal that c seeks to achieve on D.

3.2.2 Task Profiles. SANA constructs task profiles that contain ground-truth information, and uses them to synthesize idealized implementations of each ablated component (Figure 3). A profile is defined by: (Source Sequence) the sequence of gold datasets $\mathcal{D}_{\text{gold}}$ needed for each subquestion, (Sanitized Subquestions) the sequence of subquestions $\widetilde{Q}$ that have been sanitized to not leak information about the gold dataset, and (Execution Records) that specify the dataset d, the query c, the correct intent for analysis over the dataset a, and the answer A for the subquestions that require data analysis.

Example 1. *Figure 3 adapts a LakeQA task. In addition to explicitly listing the order of the three gold datasets, it removes direct mentions of dataset names in each subquestion to avoid leaking search targets. For example, a mention of* `nypd_crimes` *is rewritten as "find crime reports reported by NYPD." Subquestion 1 simply retrieves* `The Bronx` *dataset so it does not have an execution record, while the latter steps do have analysis records. The first execution record states the dataset is* `nypd-complaints`*, the correct analysis intent is to count the top-offense complaints in 2023, the correct SQL, and the answer* `18613`*.*

In SANA, we adapted tasks from LakeQA and KramaBench. The former contains all of the necessary information for the profile, and we use an LLM to remove identifying signals from each subquestion; then, we manually verified every profile to ensure each sanitized subquestion hid dataset-identifying signals without leaking the gold sources or evidence. The latter only has a question and a monolithic solution script that executes on the task's dataset, so we use an LLM to decompose this solution script into LakeQA-like sequential, dependent hops such that each hop's code is execution-verified against the intermediate answer and no subquestion spans more than one source.

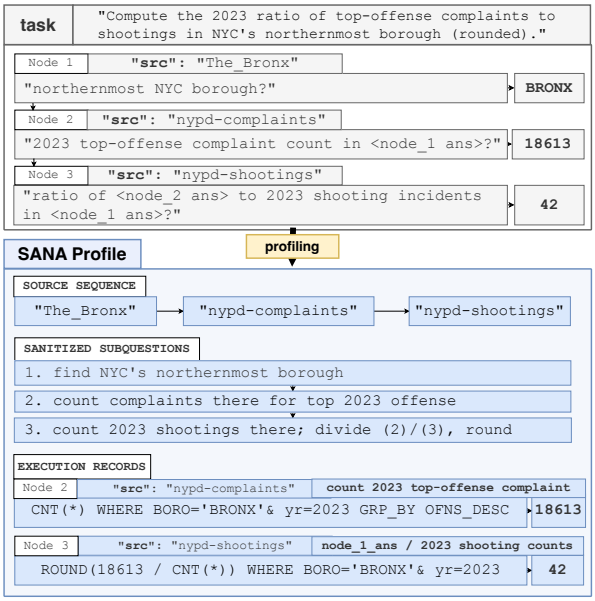

Figure 3: SANA annotates a task into source sequence, sanitized subquestions, and execution records.

3.2.3 Planning Ablation. The planning ablation isolates errors in question decomposition by providing the correct sequence of evidence needs while preserving the agent's responsibility to formulate search queries, conduct data analysis, and synthesize the final answer. In the standard setting, stronger planners produce sequenced subquestion–dataset pairs closer to the annotated sequence $((Q_1, d^1_{\text{gold}}), \ldots, (Q_K, d^K_{\text{gold}}))$.

To generate the idealized component, we provide the agent with $\tilde{Q}$ and ask it to more explicitly state the goal, suggest the tool type to use (e.g., search, query), fallback hints in case a tool call fails. For instance, "count complaints for that borough's top 2023 offense" would be rewritten as "*extract* top-*count* records of the borough in 2023 in the *NYC complaints dataset*."

3.2.4 Search Ablation. The search ablation isolates the effect of search precision: it removes irrelevant retrieval results while preserving the agent's responsibility for deciding what to search for, when to search, and when sufficient evidence has been collected. In the standard setting, the agent searches over the full data lake $\mathcal{D}$, so even when the search tool retrieves gold datasets, it may also return irrelevant datasets that increase downstream planning and analysis burden. To construct the ideal search setting, SANA replaces the standard search tool with an oracle-like operator f^{ideal} whose search space is restricted to $\mathcal{D}_{\text{gold}}$. Thus, every returned dataset is relevant to the task:

$$f^{\text{ideal}}(q, k) \subseteq \mathcal{D}_{\text{gold}} \qquad \forall q, k.$$

We implement idealized search by providing the set of gold datasets, the search query, and the search intent to a subagent that selects the datasets that match the search intent. Each dataset is augmented with metadata: an LLM-generated description [2] and a content preview. For a table, we use its schema and row samples; for a document, we use a 100-word preview. In Figure 4, "NYC complaints 2023" would return `nypd-complaints`, while "NYC farmers market" returns `dataset not found`.

3.2.5 Data Analysis Ablation. The data analysis ablation isolates implementation errors in executing the intended computation. In the standard setting, given a subquestion–dataset pair (Q_i, d_i), stronger agents are more likely to construct an analysis query that correctly derives the annotated intermediate answer A_i.

SANA replaces the standard analysis tool with $g^{\text{ideal}}_t(c_t, d_t, a_t)$, which contains an additional parameter: the agent's intent. If a_t matches with the annotated entailment a_i for each (Q_i, d^i_{gold}) pair, SANA returns the verified intermediate answer: $g^{\text{ideal}}(c_t, d_t) = A_i$, otherwise, SANA infers the computational intent from a_t and ensures c_t correctly implements it without execution failure.

We implement idealized data analysis (Figure 4) by first checking if d_t is the gold dataset. If the agent's intent is the same as the dataset's analysis intent in the execution record (via a subagent), we return the gold answer. If the intents don't match or d_t is not a gold dataset, SANA uses a stronger model (`gpt-5.4`) to generate code that matches the intent and returns the result. If the code execution fails, the model repairs the code and tries again for at most two iterations. In Figure 4, if an agent forms the right intent but filters on the wrong value case (e.g., `'Bronx'` instead of `'BRONX'`), the standard tool returns `0` while ideal returns `18613`.

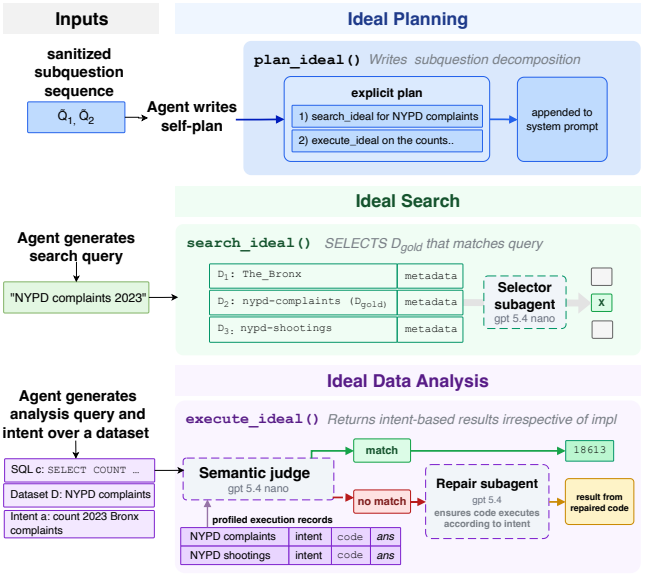

Figure 4: SANA's ideal ablation consumes one profile input and is modularized as a runtime tool.

4 EVALUATION SETUP

This section describes the task suites, ablation conditions, execution conditions, and metrics used to evaluate SANA. We follow the LakeQA evaluation setting with a budget of $B = 30$ tool calls per run and a maximum runtime of 600 seconds; each run starts from a natural-language question and ends when the agent either calls the answer-submission tool or reaches the run budget. We

evaluate two model settings: `gpt-5.4-nano` as the weaker agent and `gpt-5-mini` as the stronger agent.

4.1 Benchmarks and Tasks

We evaluate on two task suites. The first is the LakeQA `tasks_mini` subset [20], containing 135 EQA tasks. The second is the converted KramaBench [7]. To make KramaBench require data discovery, we expose its dataset as a collection of sources [25]; to align with the LakeQA evaluations with $B = 30$ tool calls, we remove the Kramabench tasks with $|\mathcal{D}_{\text{gold}}| \geq 20$ since these tasks require excessive data discovery relative to the run budget–this reduces the task count from 104 to 83.

Table 1: Benchmark statistics for SANA evaluations. Sources is the number of distinct gold sources across all tasks

Benchmark	Tasks	Sources	Avg. $\lvert D_{\text{gold}}\rvert$	Lake Size $\lvert\mathcal{D}\rvert$
LakeQA	135	499	6.9	~40 million
KramaBench-conv.	83	187	2.3	1764

4.2 Ablation Conditions

Targeted Ablations. Each SANA condition is a tuple of planning, search, and data-analysis modes, as summarized in Table 2. The targeted ablations vary one axis at a time while holding the other axes idealized. For example, when ablating planning, search and data analysis are kept idealized such that there are no conflicting bottlenecks.

Table 2: SANA condition space. Each run selects one mode from each axis.

Axis	Mode	Description
Planning	Naive	No planning tool; the agent answers directly from the question and available tools.
	Standard	Uses the planning tool, but with no sanitized subquestion sequence; the plan is entirely self-written.
	Ideal	Receives the sanitized subquestion sequence and uses the planning tool to store a plan derived from the reasoning chain.
Search	Naive	BM25 sparse lexical search over the data lake.
	Standard	Hybrid search with Reciprocal Rank Fusion (RRF), with LLM-generated table descriptions following Pneuma [2] and AutoDDG [24].
	Ideal	Ideal search tool $f^{\text{ideal}}(q, k)$ over $\mathcal{D}_{gold}$.
	Preloaded	$\mathcal{D}_{gold}$ placed directly in agent context; no search tool.
Data analysis	Standard	Writes and runs SQL or Python through the ordinary execution tools.
	Ideal	Ideal execution tool $g_t^{\text{ideal}}(c_t, d_t, a_t)$ for data analysis.

End-to-end ablations. From table 2, we also define three end-to-end conditions: (i) Naive denotes Naive Planning, Naive Search, and Standard data analysis; (ii) Standard denotes Standard planning, Standard search, and Standard data analysis; (iii) Ideal denotes Ideal planning, Ideal search, and Ideal data analysis. This comparison measures the improvement from the lower-bound baseline (Naive) to a stronger non-ideal implementation (Standard), as well as the remaining gap to the idealized upper bound (Ideal).

4.3 Execution Conditions

We vary the conditions in section 4.2 while holding the execution environment below fixed.

Execution environment. The agents are given tools for data discovery and data analysis on DuckDB or Python in an isolated sandbox on an AWS `g6.2xlarge` instance (8 vCPUs, 32 GiB RAM, NVIDIA L4 GPU), where the agent policy is orchestrated with `Strands Agent`–an open-source model-driven agent SDK that runs the reason–act–observe tool-use loop given a model, system prompt, and tools. Each tool has a 150s timeout.

Baseline difference from LakeQA. All modes use a summarizing conversation manager for context compaction as EQA tasks are often long-winded and have large intermediate query results. We also install a plugin that nudges the agent to reconsider its current strategy when the agent has repeated similar operations over 7 turns. Lastly, search tool returns the sources augmented with metadata defined in Section 3.2.4. These additions make SANA baseline stronger than the original LakeQA benchmark; thus, our results should not be interpreted as a direct reproduction of LakeQA [20].

4.4 Metrics

Our primary task-success metric is semantic match (SM). We use an LLM-as-a-judge to compare the submitted answer against the gold answer, allowing equivalent answers that differ in formatting, aliases, units, or phrasing.

To measure discovery behavior, let R be the unique sources retrieved from the search tools f_i, and A the unique sources actually accessed by the agent through data-analysis tools g_i. We report:

$$\mathrm{D}_{\text{ret}} = |\mathcal{D}_{\text{gold}} \cap R|/|\mathcal{D}_{\text{gold}}|, \qquad \mathrm{D}_{\text{acc}} = |\mathcal{D}_{\text{gold}} \cap A|/|\mathcal{D}_{\text{gold}}|,$$

the *retrieval recall* and *access recall* over the gold sources. We additionally track tool-call counts and failures in each run's log.

5 RESULTS & DISCUSSION

This section reports SANA's ablation and failure analyses on LakeQA, then tests whether they hold on the converted KramaBench.

5.1 LakeQA Ablation

Table 3: LakeQA ablation matrix (135 tasks/cell).

Model	Plan	Search	Data An.	SM (%)	D_{acc} (%)	D_{ret} (%)	Ret Tool Call	Acc Tool Call
`gpt-5.4-nano`	Naive	Ideal	Ideal	34.1	46.5	56.5	3.2	11.6
	Standard	Ideal	Ideal	31.1	47.0	57.6	4.1	13.2
	Ideal	Naive	Ideal	23.0	39.0	50.8	5.8	12.3
	Ideal	Standard	Ideal	26.7	43.4	55.4	4.7	13.2
	Ideal	Ideal	Standard	28.9	44.0	56.9	3.8	13.6
	Ideal	Ideal	Ideal	**37.0**	**47.1**	58.1	4.1	12.5
	Ideal	Preloaded	Ideal	**51.8**	**59.2**	–	0.0	17.1
`gpt-5-mini`	Naive	Ideal	Ideal	66.7	55.1	59.0	5.8	13.1
	Standard	Ideal	Ideal	66.7	56.2	58.8	5.8	14.4
	Ideal	Naive	Ideal	63.0	50.7	56.5	5.9	15.4
	Ideal	Standard	Ideal	61.5	52.4	58.0	5.8	15.9
	Ideal	Ideal	Standard	57.8	53.8	58.7	5.7	12.9
	Ideal	Ideal	Ideal	**76.3**	**56.5**	60.2	5.9	14.1
	Ideal	Preloaded	Ideal	**77.0**	**61.7**	–	0.0	16.8

5.1.1 Ablation Delta. Figure 5 reports LakeQA ablation semantic-match deltas; all three components improve performance, but by different amounts. Ideal search improves over BM25 by +14.1% (gpt-5.4-nano) and +13.3% (gpt-5-mini), and Preloaded Sources is the strongest single intervention (+28.9%, +14.1%); the intermediate BM25–Pneuma step is mixed (+3.7%, -1.5%), so non-ideal search gains remain unreliable. Ideal data analysis also yields large gains (+8.2%, +18.5%). Planning deltas are smallest (+3.0%, +9.6%), suggesting reasoning chains scale with model capability; Standard planning

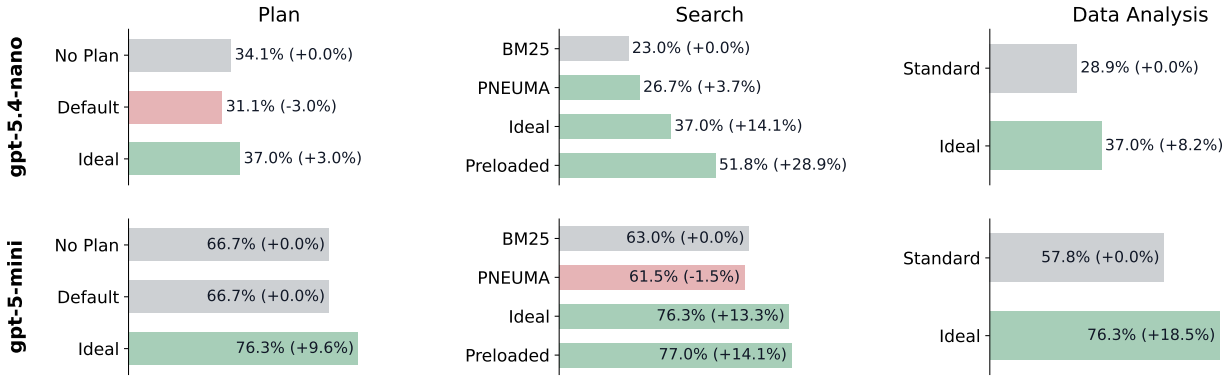

Figure 5: LakeQA ablation semantic-match delta.

gives almost no gain, as self-written plans can add unnecessary steps without preventing drift or wrong-source navigation. We analyze each component in sections 5.2.1 to 5.2.3.

Table 4: LakeQA end-to-end mode comparison ($n = 135$).

Model	Mode	SM (%)	D_{ret} (%)	D_{acc} (%)	Ret Tool Call	Acc Tool Call
gpt-5.4-nano	Naive	20.7	45.4	30.7	5.0	11.0
	Standard	19.3 (−1.5)	53.2 (+7.8)	39.5 (+8.8)	5.2	14.1
	Ideal	**37.0** (+16.3)	**58.1** (+12.7)	**47.1** (+16.4)	4.1	12.5
gpt-5-mini	Naive	56.3	53.8	45.5	7.2	12.1
	Standard	57.8 (+1.5)	56.9 (+3.1)	48.5 (+3.0)	6.0	13.2
	Ideal	**76.3** (+20.0)	**60.2** (+6.4)	**56.5** (+11.0)	5.9	14.1

5.1.2 End-to-End Mode Comparison. Table 4 compares the three end-to-end modes: Naive, Standard, and Ideal. Standard provides almost no gain over Naive, giving gpt-5-mini +1.5% and giving gpt-5.4-nano -1.5% even when D_{ret} and D_{acc} are both improved. In contrast, Ideal substantially improves both models, raising gpt-5.4-nano from 20.7% to 37.0% SM and gpt-5-mini from 56.3% to 76.3% SM. This gap shows that stronger non-ideal components do not necessarily remove EQA bottlenecks even when more data is accessed and retrieved.

5.2 LakeQA Ablation Analysis

5.2.1 Planning Ablation. Planning Idealization produces smaller gains than search or data-analysis Idealization on LakeQA. To understand why, we separate two questions: whether the agent can write a reasonable plan, and whether it actually follows that plan during the trajectory.

Plan Similarity. We first compare the plans produced under Standard and Ideal planning. A `gpt-5.4-mini` judge labels a Standard plan as similar to the Ideal plan when it preserves the same main subquestions, source path, dependency order, and core computation. By this measure, Standard plans are often close to the Ideal scaffold: 81.5% of `gpt-5.4-nano` plans and 77.8% of gpt-5-mini plans are marked similar to their corresponding Ideal plans. This suggests that agents can already create plans covering the right subquestions and intents.

Plan Trajectory. To check whether the agent actually follows their plan, we audit each trajectory against the sanitized subquestion sequence. A `gpt-5.4-mini` judge analyzes each run's agent tool-call trajectory, labelling a trajectory as Followed when the agent completes the intended ordered subproblems and Mostly Followed when the trajectory preserves the main structure but skips or detours around a minor step.

Table 5: LakeQA plan trajectory audit over targeted plan ablations.

Agent	Plan	Followed	Followed+Mostly
gpt-5-mini	Naive	21.2%	49.6%
	Standard	12.6%	46.4% (−3.2)
	Ideal	25.9%	56.5% (+6.9)
gpt-5.4-nano	Naive	5.7%	19.0%
	Standard	8.4%	24.0% (+4.9)
	Ideal	7.9%	28.1% (+9.1)

The trajectory audit clarifies why ideal planning has limited impact. gpt-5.4-nano is especially poor at following plans: only 19.0% of trajectories are at least Mostly Followed at baseline, and ideal only raises it to 28.1%. gpt-5-mini follows plans more often, but the gain from ideal planning is still modest (49.6% to 56.5%). Thus, simply providing the correct decomposition does not mean agents follow that decomposition reliably. These results suggest that agents are more bottlenecked by their ability to follow plans than their ability to decompose questions.

5.2.2 Search Ablation. Figure 5 separates two search-related bottlenecks: retrieval quality and search navigation. Ideal search improves over BM25 for both models, showing that retrieving the right sources from a large data lake is a major bottleneck. However, Preloaded Sources improves over Ideal Search substantially on gpt-5.4-nano while barely affecting gpt-5-mini. This suggests that weaker agents also struggle at search navigation: formulating the right search queries, deciding when to search, and continuing using the right sources once they are available. In contrast, the stronger model gains little improvement from removing search entirely.

Search ablation therefore exposes two separate needs: better retrieval infrastructure and better agent policies for query formulation, source commitment, and evidence collection.

5.2.3 Data Analysis Ablation. Ideal data analysis helps gpt-5-mini more than gpt-5.4-nano on LakeQA (Figure 5). This suggests that gpt-5-mini often reaches the right source context and computation intent, but fails to implement the operation correctly in SQL or Python. The smaller gain for gpt-5.4-nano does not mean that execution is less important for weaker models: rather, many gpt-5.4-nano failures occur before data analysis implementation becomes a bottleneck: choosing the wrong source, scope, or analysis intent.

Table 6: Failure families in LakeQA traces. Entries are event shares; Meaning lists (gpt-5-mini, gpt-5.4-nano) subgroups.

Group	gpt-5-mini	gpt-5.4-nano	Meaning
Task/planning failures	7.4%	11.3%	Reasoning chain divergence (5.4%, 8.7%); question constraint misread (1.9%, 2.6%).
Wrong source target failures	0.0%	7.6%	Chose the wrong dataset, table, source family, or source version.
Execution/computation failures	39.6%	26.2%	Computation intent failures: wrong scope/filter (30.9%, 20.1%); computation/aggregation error (4.9%, 2.8%); extraction/parsing error (3.9%, 3.2%).
Incomplete evidence failures	12.2%	12.3%	Ran out of budget before gathering enough evidence (10.3%, 5.0%); submitted early with incomplete evidence (1.9%, 7.3%).
Turn-waste failures	2.1%	8.8%	Repeated actions without progress: query execution/repair loop (0.8%, 5.5%); schema inspection loop (0.2%, 1.8%); low-yield search loop (0.6%, 1.4%); same-hop repetition even after gathering enough evidence (0.6%, 0.2%).
Finalization failures	21.0%	13.2%	Correct evidence or computed value appeared, but the submitted answer was wrong.
Tool blocker failures	17.7%	20.6%	Files, tools, repair calls, unsupported formats, or runtime limits blocked progress.

Since ideal execution preserves the agent's stated intent, it fixes implementation errors but not wrong requests. Thus, data analysis ablation mainly separates code-generation failures from upstream source-selection and intent errors.

5.3 LakeQA Failure Analysis

In this section, we audit failed traces and categorize where every evaluation run with an incorrect semantic outcome fails.

Audit Method. Each failed run is reviewed using a two-stage audit with `gpt-5.4-mini`. First, the auditor reads the task, the SANA profile, evaluation results, and log trace, then extracts a compact evidence trace of all failures and the turns at which they occur. Then, a separate labeler assigns one or more failure events from the taxonomy in Table 6 using only that evidence trace, such that a single failed run may contribute multiple failures such as answer finalization and planning failures. Since co-occurring symptoms are recorded together regardless of root cause (e.g., misreading question constraint and then filtering the wrong rows returns both constraint misread and wrong scope/filter labels), we treat these LLM-assisted labels as diagnostic rather than ground truth.

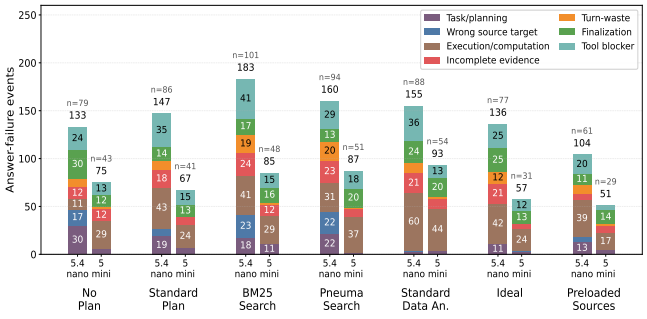

Figure 6: Answer-failure groups across LakeQA ablations. Bold numbers are total events and *n* the failed runs per bar.

Figure 6 and Table 6 explain why the two models respond differently to the same conditions. Gpt-5.4-nano produced 1018 failure events over 586 failed runs and gpt-5-mini 515 over 297, about 1.74 events per failed run for both models. Table 6 gives the event distribution within each model and Figure 6 the condition-level counts. The distributions differ: gpt-5.4-nano's failures spread across all groups, while gpt-5-mini's concentrate in execution/computation (57.2% of its failed runs) and finalization (35.7%).

The clearest gap is source scoping: gpt-5-mini has 0% wrong-source failure labels whereas gpt-5.4-nano receives 12.8%. This matches fig. 5, where gpt-5.4-nano benefits more from Ideal Search and Preloaded Sources as they prevent wrong-source branches. Gpt-5.4-nano's other disproportionate failures—turn-waste (8.8% gpt-5.4-nano vs. 2.1% gpt-5-mini), reasoning-chain divergence (8.7% vs. 5.4%), and early submission with incomplete evidence (7.3% vs. 1.9%)—are trajectory failures, agreeing with Subsection 5.2.1 where gpt-5.4-nano is worse at staying on its plan.

Idealizing search (BM25 to Ideal in Figure 6) reduces gpt-5.4-nano's wrong-source (23 to 0), turn-waste (19 to 12), and divergence (18 to 11) events, partly because failed runs themselves drop (101 to 77). For gpt-5-mini these are already near zero and idealized search mainly trims incomplete-evidence errors, suggesting the stronger model already selects sources and operations well while the weaker one still struggles.

Planning is more selective, helping mainly the smaller model read question constraints and finalize answers. Finalization, computation intent, and turn budget remain failure modes for both, even under Ideal and Preloaded where source access is solved. Overall the two models fail differently: gpt-5-mini usually reaches the right evidence but mis-executes or mis-finalizes, while gpt-5.4-nano's failures spread across source targeting, plan divergence, turn-waste, and premature stopping, reflecting weaker trajectory control. The ablation gains thus reflect differences in action policies, not only component quality.

5.4 KramaBench Ablation

Table 7: KramaBench ablation matrix (83 tasks/cell).

Model	Plan	Search	Data An.	SM (%)	D_{acc} (%)	D_{ret} (%)	Ret Tool Call	Acc Tool Call
gpt-5.4-nano	Naive	Ideal	Ideal	75.9	81.7	89.9	1.7	3.2
	Standard	Ideal	Ideal	85.5	89.3	91.0	1.4	3.7
	Ideal	Naive	Ideal	85.5	50.1	78.9	2.9	4.0
	Ideal	Standard	Ideal	90.4	43.5	77.1	2.1	4.0
	Ideal	Ideal	Standard	63.9	92.9	93.9	1.3	2.9
	Ideal	Ideal	Ideal	**87.9**	**92.3**	95.8	1.6	3.5
	Ideal	Preloaded	Ideal	**91.6**	**98.4**	–	0.0	3.7
gpt-5-mini	Naive	Ideal	Ideal	86.8	93.9	95.8	1.4	4.6
	Standard	Ideal	Ideal	89.2	93.7	94.3	1.4	4.6
	Ideal	Naive	Ideal	84.3	51.6	82.0	2.7	4.2
	Ideal	Standard	Ideal	90.4	50.8	78.9	2.0	4.1
	Ideal	Ideal	Standard	75.9	96.7	96.7	1.5	3.4
	Ideal	Ideal	Ideal	**90.4**	**97.0**	97.6	1.2	4.1
	Ideal	Preloaded	Ideal	**92.8**	**99.0**	–	0.0	4.6

5.4.1 Ablation Delta. Figure 7 shows a different profile from LakeQA. Data analysis is the largest KramaBench intervention, improving gpt-5.4-nano by +24.1% and gpt-5-mini by +14.5%, indicating that KramaBench is analysis-heavy. Search is less limiting: Ideal search improves over BM25 by +2.4% for gpt-5.4-nano and +6.0% for gpt-5-mini, while Preloaded Sources adds +6.0% and +8.4%. This likely reflects the benchmark construction: retrieval is artificially introduced by converting KramaBench into a LakeQA-style task, while the underlying tasks remain centered on data-to-insight execution. Planning helps gpt-5.4-nano more than gpt-5-mini, suggesting that weaker models benefit more from explicit decomposition on KramaBench.

5.4.2 End-to-End Mode Comparison. Table 8 shows that Standard improves over Naive on KramaBench, especially for gpt-5.4-nano

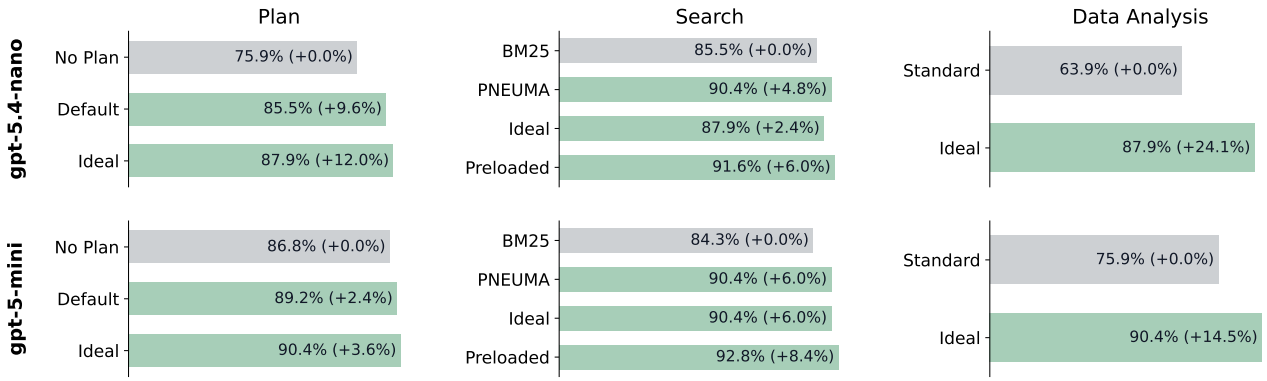

Figure 7: KramaBench ablation semantic-match delta.

Table 8: KramaBench end-to-end mode comparison ($n = 83$).

Model	Mode	SM (%)	D_{ret} (%)	D_{acc} (%)	Ret Tool Call	Acc Tool Call
gpt-5.4-nano	Naive	44.6	64.2	31.8	3.2	3.3
	Standard	57.8 (+13.3)	70.6 (+6.4)	44.6 (+12.8)	2.4	3.6
	Ideal	**87.9** (+43.4)	**95.8** (+31.6)	**92.3** (+60.4)	1.6	3.5
gpt-5-mini	Naive	62.6	64.7	36.0	3.8	3.6
	Standard	66.3 (+3.6)	68.5 (+3.9)	34.8 (−1.2)	3.2	3.8
	Ideal	**90.4** (+27.7)	**97.6** (+32.9)	**97.0** (+61.1)	1.2	4.1

(+13.3%). Ideal produces the largest gains, reaching 87.9% SM for nano and 90.4% SM for gpt-5-mini. Under Ideal, retrieval recall reaches 95.8%–97.6% and access recall reaches 92.3%–97.0%, so the remaining errors are unlikely to come mainly from missing sources.

6 CONCLUSION AND LIMITATIONS

Conclusion. SANA asks a diagnostic question: when an EQA agent fails over a data lake, which part of the runtime is responsible? By varying only the planning, search, and data-analysis interfaces across LakeQA and the converted KramaBench, we find that no single component explains EQA failure. Data analysis is a consistent bottleneck on both benchmarks; search dominates on LakeQA, where agents must navigate a large lake, but is less dominant on KramaBench, whose data-to-insight tasks with fewer sources shift the burden to analysis; and planning is the weakest intervention, as agents produce plans close to the gold decomposition but do not reliably follow them. SANA thus does not rank a component as universally hardest, but exposes how bottlenecks shift with benchmark structure. Even after idealizing every component, however, failures persist—residual agent-policy failures that these ablations cannot remove.

The LakeQA failure audit dissects this residual gap: smaller agents fail broadly across source choice, progress control, incomplete evidence, tool blockers, and finalization. Stronger agents are better at staying within the right source context, but still fail during computation and final-answer synthesis.

These results suggest future EQA systems need more than better retrievers or code generation; they need a stronger action policy that acts on the current trajectory. A source-commitment mechanism should lock the validated working set and require justification to switch, since weaker agents branch to wrong sources. A plan-adherence mechanism should gate each tool call on an active subgoal and replan when successive calls fail to advance it, since agents plan well but drift during execution. An intent check should validate a computation's scope, filters, and units against the question before execution, since idealization corrects implementation but not wrong requests. A submit-time verifier should re-derive the answer from logged evidence and block submission on mismatch or uncovered subquestions. Because bottlenecks shift with the model and task, these mechanisms should target the diagnosed weakness rather than serve as a generalizable fix.

Limitations.

(i). Diagnostic idealization. SANA's ideal tools consume ground truth, so they are optimistic upper bounds unattainable in practice. The measured gaps thus overestimate attainable gains and, since each ablation holds the other axes ideal, are isolated contributions, not marginal effects when components are jointly non-ideal. The ranking may shift with practical implementations.

(ii). Lack of alternative valid sources. Each task has a single annotated $\mathcal{D}_{\text{gold}}$, so alternatives are neither surfaced by ideal search nor shortcut by ideal computation (which falls back to repaired code execution). This underrepresents D_{ret}, D_{acc}, and the measured impact of the data-analysis ablation. Enumerating alternatives is a benchmark-generation effort beyond SANA's scope.

(iii). Judge-based measurements. Semantic-match, plan-trajectory, and failure-audit labels rely on LLM judges. These make evaluation practical and credit semantically equivalent answers, but can introduce false positives or false negatives.

(iv). Task-suite scope. We evaluate only LakeQA and a LakeQA-converted KramaBench, so our findings are evidence about these data-lake settings, not a universal ranking of search, planning, and data-analysis difficulty across agent benchmarks.

ACKNOWLEDGMENTS

This research received funding from DAPLab corporate support in the form of funding and/or compute from Amazon, IntellectAI, Infosys, Tidalwave, Veris, Shopify, Microsoft, Thinking Machines, Dandy, Perplexity, and Daytona. The views and conclusions presented here are those of the authors and should not be interpreted as representing the official positions of the funding organizations.

REFERENCES

[1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=hSyW5go0v8

[2] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging LLMs for Tabular Data Representation and Retrieval in an End-to-End System. *Proc. ACM Manag. Data* 3, 3, Article 200 (June 2025), 28 pages. https://doi.org/10.1145/3725337

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[4] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. 2021. Open Question Answering over Tables and Text. In *International Conference on Learning Representations*. https://openreview.net/forum?id=MmCRswl1UYl

[5] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Shaohan Wang, Pengyu Wang, Xiaorui Wang, and Zhendong Mao. 2026. A-RAG: Scaling Agentic Retrieval-Augmented Generation via Hierarchical Retrieval Interfaces. *arXiv preprint arXiv:2602.03442* (2026).

[6] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-Aware Dataset Discovery from Data Lakes with Contextualized Column-Based Representation Learning. *Proc. VLDB Endow.* 16, 7 (March 2023), 1726–1739. https://doi.org/10.14778/3587136.3587146

[7] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Mike Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. 2026. KramaBench: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=fZfUdeCC5X

[8] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=XmProj9cPs

[9] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=dI4wzAE6uV

[10] Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. 2026. Beyond Semantic Similarity: Rethinking Retrieval for Agentic Search via Direct Corpus Interaction. arXiv:2605.05242 [cs.IR] https://arxiv.org/abs/2605.05242

[11] Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop Reading Comprehension through Question Decomposition and Rescoring. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 6097–6109. https://doi.org/10.18653/v1/P19-1613

[12] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2022. FeTaQA: Free-form Table Question Answering. *Transactions of the Association for Computational Linguistics* 10 (2022), 35–49. https://doi.org/10.1162/tacl_a_00446

[13] Thang Nguyen, Peter Chin, and Yu-Wing Tai. 2025. Ma-rag: Multi-agent retrieval-augmented generation via collaborative chain-of-thought reasoning. *arXiv preprint arXiv:2505.20096* (2025).

[14] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 36339–36348. https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf

[15] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=dHng2O0Jjr

[16] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=Yacmpz84TH

[17] Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136* (2025).

[18] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics* 10 (2022), 539–554. https://doi.org/10.1162/tacl_a_00475

[19] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 10014–10037. https://doi.org/10.18653/v1/2023.acl-long.557

[20] Haonan Wang, Jiaxiang Liu, Yurong Liu, Austin Senna Wijaya, Tianle Zhou, Eden Wu, Yijia Chen, Wanting You, Reya Vir, Daniela Pinto Veizaga, Grace Fan, Yusen Zhang, Juliana Freire, and Eugene Wu. 2026. LakeQA: An Exploratory QA Benchmark over a Million-Scale Data Lake. In *Forty-third International Conference on Machine Learning*. https://openreview.net/forum?id=gWoEOFW0CR

[21] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 2369–2380. https://doi.org/10.18653/v1/D18-1259

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

[23] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[24] Haoxiang Zhang, Yurong Liu, Aécio Santos, Wei-Lun (Allen) Hung, and Juliana Freire. 2026. AutoDDG: Automated Dataset Description Generation using Large Language Models. *Proc. ACM Manag. Data* 4, 1, Article 12 (April 2026), 27 pages. https://doi.org/10.1145/3786626

[25] Jiani Zhang, Sercan O. Arik, Cosmin Arad, Fatma Ozcan, and Alon Halevy. 2026. An Agentic Approach to Metadata Reasoning. arXiv:2604.20144 [cs.DB] https://arxiv.org/abs/2604.20144

[26] Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 3277–3287. https://doi.org/10.18653/v1/2021.acl-long.254