

L.A.K.E.: Logic Agent for Knowledge Extraction in Data Planning

Jean-Flavien Bussotti, Naoki Otani, Eser Kandogan
{jflavien,naoki,eser}@megagon.ai

Abstract

Data lakes in modern enterprises are massive, heterogeneous, and noisy, often preventing non-experts from effectively extracting value. Bridging the semantic gap between ambiguous user intent and explicit data requires orchestrating multiple tools under an open-world assumption. However, reliably executing these compound AI workflows to solve complex knowledge extraction tasks, while also providing the transparency needed for evaluation and debugging, remains a significant bottleneck. We propose **L.A.K.E.** (Logic Agent for Knowledge Extraction), an agentic data planning framework designed to map natural language questions to executable workflows over diverse data sources. Rather than relying on a brittle “one-size-fits-all” approach, L.A.K.E. dynamically generates a declarative plan comprised of modular operators—spanning relational and semantic functions over heterogeneous data sources. Within this framework, we introduce and benchmark three distinct planning regimes: *Iterative Planning*, *Single-Shot Tree Planning*, and *Cascade Planning*. We present an interactive demonstration platform that enables users to visually compare the latency and robustness trade-offs of these planners. By rendering execution paths as interactive Directed Acyclic Graphs (DAGs) with step-level provenance, L.A.K.E. provides the critical observability needed to establish trust, debug failures, and optimize data planning for enterprise-scale lakes.

CCS Concepts

• **Information systems;**

Keywords

Data agents, agentic data workflows, data planning, metadata, text-to-SQL, enterprise data lakes

ACM Reference Format:

Jean-Flavien Bussotti, Naoki Otani, Eser Kandogan . 2026. L.A.K.E.: Logic Agent for Knowledge Extraction in Data Planning. In *ACM Conference on AI and Agentic Systems (CAIS '26)*, May 26–29, 2026, San Jose, CA, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3786335.3813202>

1 Introduction

Enterprise data lakes have become increasingly complex: even specialists struggle to manage and retrieve data efficiently. Business analysts and end-users—who are often unfamiliar with data query languages and/or schema—face particularly high barriers to extracting value from available data assets. In this context, we present a flexible framework for data planning that explicitly centers user intent, heterogeneous resources, and open-world uncertainty.

Our motivation is threefold:

First, **enterprise data is massive, heterogeneous, and noisy**. Standard text-to-SQL benchmarks such as Spider [18] operate under a closed-schema, single-database assumption. However, realistic enterprise environments involve complex, unnormalized databases with hundreds of tables, undocumented relationships, and ambiguous naming conventions. The data itself is often plagued by missing values, inconsistent formatting, and cross-system dependencies [2, 11]. Enterprise information is notoriously fragmented across relational databases (e.g., PostgreSQL), semi-structured stores (e.g., MongoDB), and unstructured text (e.g., documentation of business logic), exposing the limitations of the closed-world assumption prevalent in existing NL2SQL systems. Modern data agents must treat these diverse modalities, and even external resources such as the web and LLMs themselves, as first-class data sources with their own query languages and constraints [1, 9].

Second, a persistent **semantic gap and evolving user demands** complicate the handling of user queries. The distance between implicit user *intent* and explicit *data* is often large, as databases generally lack common-sense and the proprietary, company-specific context necessary to correctly interpret information. Bridging this gap frequently requires external domain knowledge, implicit assumptions, or derived features that are not directly encoded in rigid schemas [2, 11]. Furthermore, users increasingly require task-centric interactions (e.g., “Find me internal candidates in the Bay Area with at least three years of ML experience who are ready for a leadership role”) rather than simple data retrieval. This shift calls for “compound AI” systems [5] capable of schema-agnostic reformulation and orchestrating multiple tools in a coordinated manner, rather than issuing a single SQL statement.

Finally, **observability and planning evaluation** remain significant bottlenecks. Recent analyses of LLM agents in production reveal that developers heavily favor simple, controllable workflows over unbounded autonomy, identifying reliability and correctness as their primary hurdles; notably, nearly 70% of production agents are capped at 10 steps or fewer before requiring human intervention to prevent cascading failures [12]. Furthermore, because academic benchmarks rarely capture bespoke enterprise data, evaluating these systems currently relies heavily on manual, human-in-the-loop inspection. Consequently, developers need transparency to understand how an agent decomposes a task, selects tools, and handles errors. Much like the EXPLAIN command in relational databases establishes trust and highlights optimization opportunities for SQL execution, the industry requires an interactive platform to visually inspect, debug, and empirically compare different planning regimes prior to deployment.



This work is licensed under a Creative Commons Attribution 4.0 International License. CAIS '26, San Jose, CA, USA

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2415-2/2026/05
<https://doi.org/10.1145/3786335.3813202>

In this work, we present L.A.K.E. (Logic Agent for Knowledge Extraction)¹, a library that connects user questions to heterogeneous data sources by providing specialized planning strategies over a shared operator set. We focus on limiting the search space of planning while still allowing rich compositions of Large Language Models (LLMs) with execution engines. Concretely, we ask:

How can we design a framework that not only maps user intent to workflows over heterogeneous data but also provides the observability—akin to a database query plan—needed to establish trust, inform optimizations, and evaluate the trade-offs between different planning regimes?

To this end, our L.A.K.E. framework implements three planners—Iterative Planning, Single-Shot Tree Planning, and Cascade Planning—on top of the Blue Architecture framework for agentic enterprise workflows [5].

See video demonstration on <https://youtu.be/c0wpeVFwWwI>
Code : <https://github.com/megagonlabs/LAKE>

2 Related Work

Our work is situated at the intersection of text-to-SQL, open-domain question answering, and agentic planning for enterprise data systems. Traditional Text-to-SQL systems and benchmarks [7, 18] predominantly assume a closed-world, single-database setting. However, enterprise environments demand an *open world assumption* where data is fragmented across heterogeneous systems [10]. To address this, compound AI systems have begun integrating NL2SQL with retrieval-augmented generation (RAG) [9] and iterative agentic refinement [4]. Concurrently, declarative frameworks like Palimpzest [8] and LOTUS [13] embed LLMs directly into data pipelines as semantic operators to optimize execution. Yet, these execution-focused engines largely assume the user can already provide a well-defined query plan.

In contrast, L.A.K.E. addresses the upstream challenge of *planning*: dynamically resolving ambiguous user intent into executable operator graphs. While recent architectures advocate for modular planning paradigms (e.g., Chain-of-Thought [15], ReAct [16], Plan-and-Solve [14]) and enterprise orchestration (e.g., Blue [5]), selecting the optimal strategy remains difficult. L.A.K.E. fills this gap by providing an interactive, observable demo platform to compare distinct planning regimes—iterative loops, single-shot tree, and cascade—over a shared operator set. Our work is complementary to general agent frameworks (e.g., LangChain-, or LlamaIndex-based agents [3, 9]), but focuses on planning over a constrained operator set and empowering developers to empirically evaluate latency and robustness trade-offs prior to deployment.

3 Method

L.A.K.E. is implemented as a planning and execution layer atop the Blue Architecture [5]. The system exposes multiple planning strategies that generate a directed acyclic graph (DAG) of executable steps over a shared toolset. Supported data sources span structured

(e.g., PostgreSQL), semi-structured (e.g., MongoDB), unstructured, and other modalities like vector stores and web search.

Empirical experiments revealed strict trade-offs between latency, cost, and robustness [6], motivating L.A.K.E.’s multi-strategy approach. *Iterative Planning* maximizes robustness on ambiguous tasks by dynamically adapting to intermediate data; *Single-Shot Tree Planning* minimizes LLM calls for low-latency execution on predictable queries; and *Cascade Planning* offers a modular middle ground by decoupling semantic intent from operator linking.

These distinct architectures offer valuable operational advantages. Upfront planners (Single-Shot and Cascade) generate explicit DAGs, enabling human-in-the-loop review prior to execution. By decoupling intent from execution, Cascade’s modular design curtails the compounding effect of LLM hallucinations and allows simpler sub-tasks to be routed to cost-effective models. Furthermore, Cascade accelerates overall latency via pipelining: the engine links and executes initial steps concurrently while downstream steps are still being prepared. Further details regarding the tools and the evaluation of the planners can be found in Appendix B.

3.1 Iterative Planning

The Iterative Planner utilizes a common function-calling pattern characteristic of modern LLM agents [17]. Operating under an agentic, one-step policy, the LLM is given the tool set and data description, and it proposes exactly one concrete execution step at a time. In this framework, each step maps directly to a tool in our shared operator set (e.g., issuing a specific SQL query or executing a search) or a termination signal, rather than a plan-level meta-operator that modifies an overarching DAG.

The selected step is executed, and a compact summary of the result is fed back to the LLM. This loop continues until a final answer or explicit failure is produced.

While this approach is theoretically less efficient due to its stop-and-wait nature, it allows for dynamic adaptation based on intermediate results and the discovery of new constraints or opportunities during execution [16].

Figure 1 provides an overview of the Iterative Planner.

3.2 Single-Shot Tree Planning (an extension of plan and execute)

In Single-Shot Tree Planning, the LLM is tasked with producing a runnable plan directly from the user question. The output is a JSON object that contains the model’s reasoning steps and a nested tool tree.

To evaluate how model capability and prompt structure influence this process, we introduce two primary structural approaches and model classes:

- **Single-Shot Tree Planning (guided NLMerge):** The model (e.g., GPT-4o class) follows an explicit checklist enforced by the prompt (e.g., *Data Availability* → *Reformulation* → *Tool Selection*). Within this guided approach, we define two generation directions for constructing the nested JSON: building the tree from the leaves to the root (*inner* → *outer*) versus root to leaves (*outer* → *inner*). The generated tree is then parsed and executed asynchronously by the engine.

¹We use "knowledge extraction" here to denote the end-to-end retrieval and synthesis of answers from diverse enterprise data sources, broadly encompassing more than just the creation of knowledge in a standardized form.

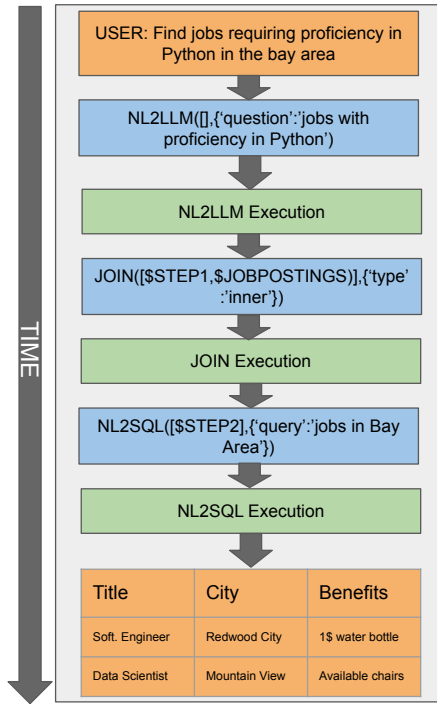


Figure 1: Iterative planning pipeline, where the planner repeatedly chooses the next action based on prior steps and summarized memory.

- **Single-Shot Tree Planning (reasoning):** The checklist is omitted entirely. Instead, a stronger, reasoning-optimized model (e.g., GPT-5 class) relies on its internal reasoning heuristics to deduce the full task sequence. While the engine still executes the final emitted plan, the advanced model functionally acts as a **monolithic planner**, requiring no external prompting guardrails to formulate complex nesting.

Figure 2 provides an overview of the Single-Shot Tree Planning.

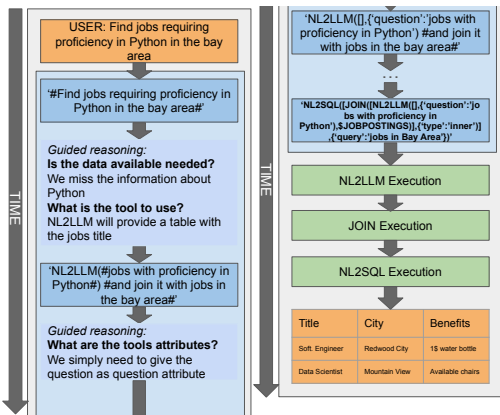


Figure 2: Single-Shot Tree planning pipeline, where the LLM directly emits a runnable tool tree as JSON.

3.3 Cascade Planning

In Cascade Planning, the system first constructs a high-level, natural-language plan as a *linear* sequence of steps. The planner outputs, for each step, only the tool name and an informal intent description. A separate **Linker** then converts each natural-language step into a concrete, executable operator invocation, binding its inputs to outputs from previous steps. Once the linking process is finalized, we obtain a DAG constructed from the individual dependencies of the steps, which enables the engine to determine execution order and exploit parallelism. Phase 1 is user-focused focusing on intent, while Phase 2 is data/operator-focused.

This approach offers two main benefits:

- (1) **Potential for rapid execution.** In theory, execution can begin as soon as the first step has been linked, without waiting for the entire plan to be fully resolved into a DAG. However, as discussed in our experiments, this speed advantage can be offset if downstream refinement loops are triggered frequently.
- (2) **Intent-execution decomposition.** It separates the “what” (semantic intent) from the “how” (concrete linking of inputs and attributes), which simplifies prompts and allows for different models or prompt regimes for planning and linking.

Cascade Planning further incorporates a **Refiner** that can be invoked at each element of the pipeline. While an upfront planner may fail to account for data granularity or implicit constraints, the refiner can leverage execution feedback to detect anomalies and adjust. For example, if a user requests “jobs in the Bay Area” but the database only stores city-level locations, a naive query such as `Location = 'Bay Area'` returns no rows. The refiner detects the empty result, infers a granularity mismatch, and triggers a mitigation strategy, such as rewriting the query to include a set of Bay Area cities.

Figure 3 provides an overview of the Cascade Planner.

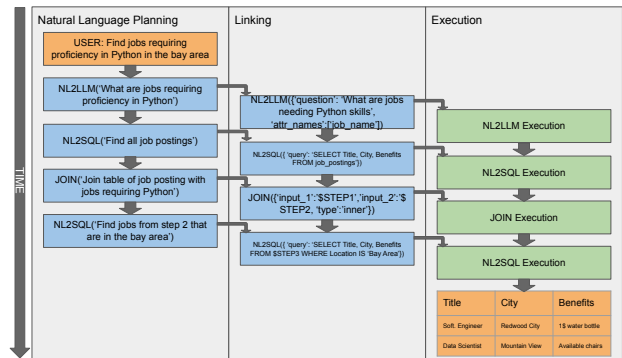


Figure 3: Overview of the Cascade Planner pipeline, illustrating linear plan generation, linking, and refinement over a DAG of operators.

4 Demonstration

To support the evaluation of our system, we provide an interactive web interface designed to make the system’s behavior fully observable. The interface allows users to (i) enter a natural-language question, (ii) select a subset of available tools, (iii) choose a planning variant, and (iv) inspect the resulting tool pipeline step-by-step.

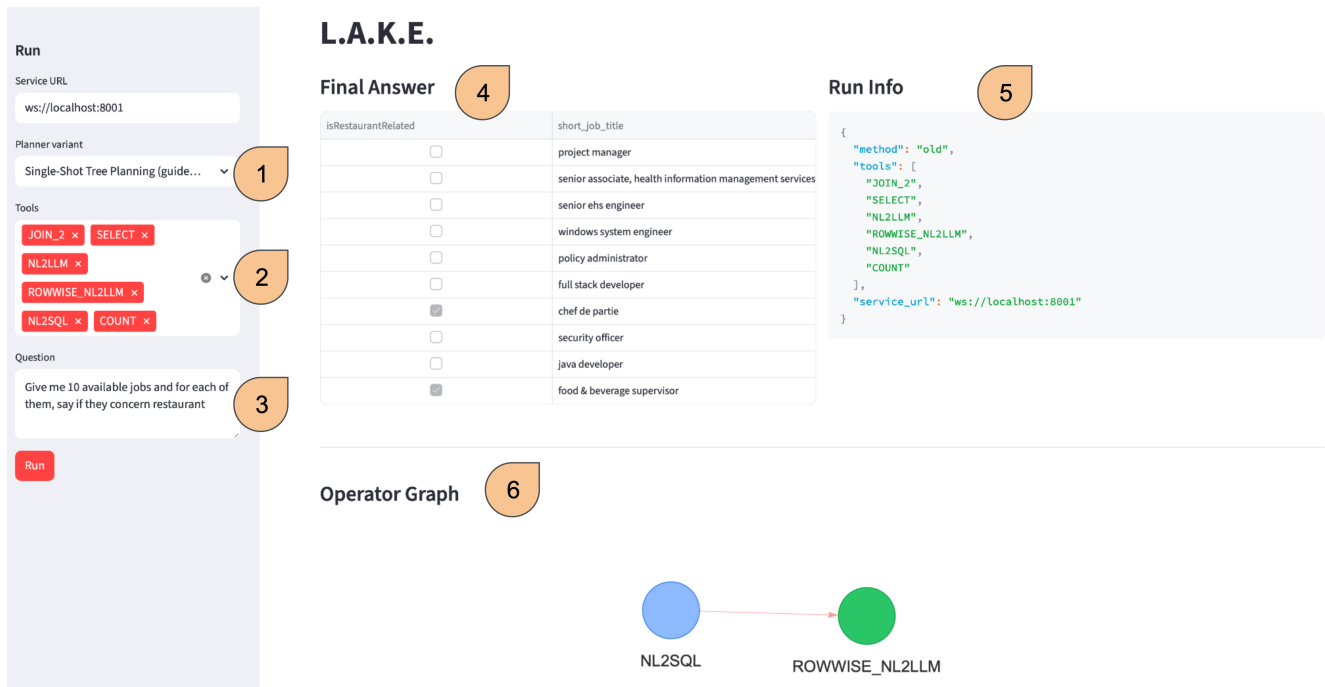


Figure 4: The L.A.K.E. demonstration interface. The left pane features (1) planner selection, (2) tool pool management, and (3) a natural language query input. The right pane provides deep observability via (4) the final result table, (5) the specific execution configuration, and (6) a DAG visualizing tool dependencies and data linking.

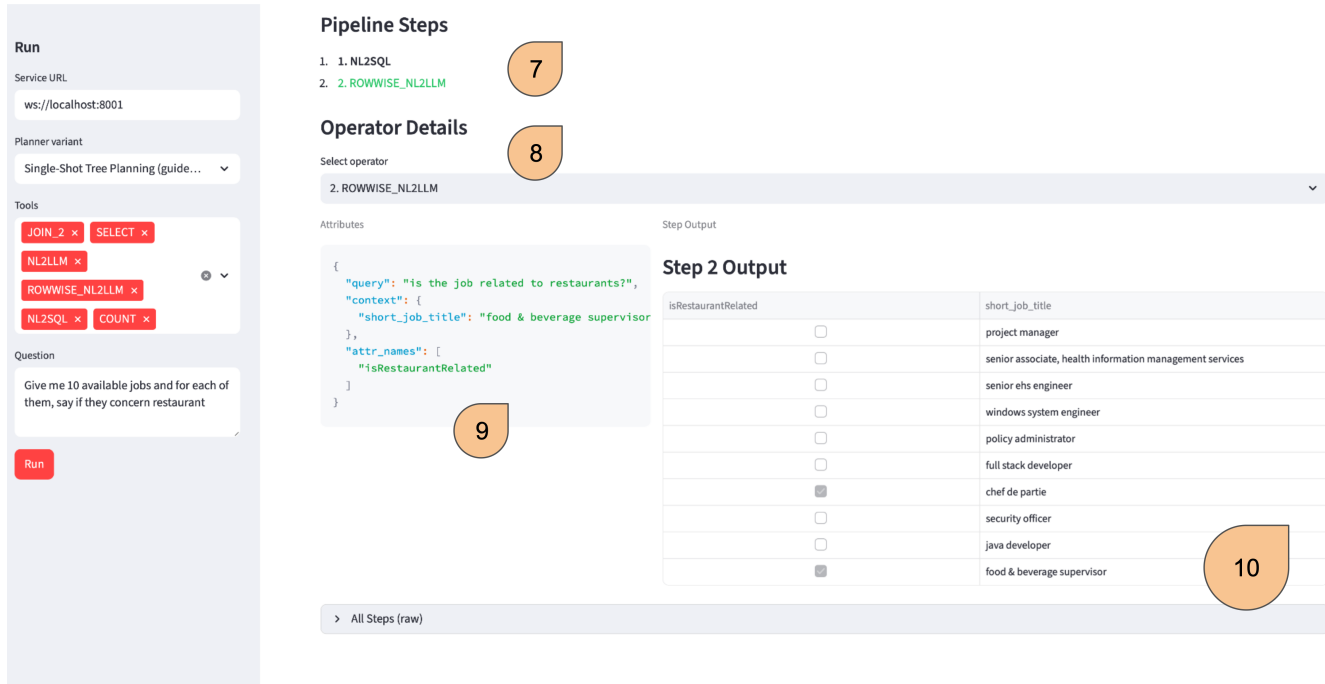


Figure 5: Detailed inspection view and step-level provenance. (7) A summary of the executed pipeline steps. (8) An operator selector for granular inspection, which updates the view to show (9) specific attributes and parameters used during execution and (10) a tabular display of the intermediate results.

Beyond the final answer, the interface exposes intermediate artifacts—such as tool attributes and outputs—and the structure of the generated plan as a DAG.

4.1 User Workflow

The interface follows a four-stage workflow to guide the user from query to inspection:

- (1) **Specify the task:** The user enters a free-form natural language query (e.g., “Give me 10 available jobs and for each of them, say if they concern restaurant”).
- (2) **Constrain the planner:** The user selects the specific tools available for the run (e.g., NL2SQL, ROWWISE_NL2LLM, JOIN, SELECT, or COUNT). This enables controlled experiments, such as ablations, to highlight specific planner behaviors.
- (3) **Select a planner variant:** The user chooses a planning method (e.g., single-run vs. iterative) from the available variants.
- (4) **Run and inspect:** Upon execution, the UI triggers the pipeline execution code, collects a structured JSON payload, and renders a complete, inspectable trace of the run.

4.2 Operator Graph (Plan DAG) Visualization

A central element of the demonstration is the visualization of the computed plan DAG. In this view, each node represents an operator or tool invocation, while edges represent dataflow dependencies. This visualization serves two primary purposes:

- **Explainability:** It makes the planner’s decomposition explicit, showing how sub-questions were created, which tools were chained, and how intermediate results propagate.
- **Debugging and Analysis:** It provides insight into failure modes, such as missing attributes, unexpected intermediate data shapes, or downstream operators consuming incorrect fields.

The UI includes an interactive graph component allowing users to click a node to focus on specific execution details. For environments without graph dependencies, the UI provides a fallback dropdown selection system.

4.3 Step-level Provenance

To ensure deep observability, the interface emphasizes step-level provenance for every operator in the pipeline:

- **Attributes:** For any selected operator, the UI displays the parameters used for execution, such as the generated SQL query, database identifiers, or LLM prompts.
- **Outputs:** The UI captures and renders the output produced at each step. Given that many tools return tabular data, the interface detects these structures and renders them as interactive tables.

Additionally, the interface provides an ordered “Pipeline Steps” list for quick navigation and an expandable raw JSON view of all recorded steps for logging and reproducibility.

4.4 Reproducibility and Deployment

The demonstration system is designed for both automated experimentation and live interaction:

- **CLI Runner:** A command-line entry point produces the full JSON payload for any configuration, enabling scripted experiments and artifact collection.
- **Web UI:** A Streamlit-based application provides an interactive layer over the execution path, suitable for live demonstrations and generating figures for qualitative analysis.

The full interface can be seen in Figure 4 and 5.

5 Conclusion and Future Work

We have presented L.A.K.E., a library designed to bridge the gap between complex enterprise data lakes and non-expert user intent through modular planning strategies. By implementing Iterative planning, Single-Shot Tree, and Cascade regimes, we provide a flexible framework for orchestrating heterogeneous data tools under an open-world assumption. As demonstrated, the ability to decompose user intent into executable DAGs allows for a transparent and explainable path from raw query to final answer.

Our work highlights that the choice of planning strategy—whether prioritized for robust self-correction (Iterative), rapid low-latency efficiency (Single-Shot Tree), or modular intent-execution decoupling (Cascade)—is a critical design decision in enterprise data systems. We include limitations in Appendix A. Future work will focus on:

- (1) **Rule-based planning.** Integrating rule-based components to systematically handle recurrent input/output mismatches discovered during experiments (e.g., automatically simplifying joins or enforcing schema guards).
- (2) **Human-in-the-loop verification.** Enhancing the demonstration interface to allow users to intervene and refine the planning DAG before or during execution.
- (3) **Dynamic strategy routing.** Developing a “meta-planner” that predicts the optimal planning regime for a given query to balance latency, cost, and expected success.
- (4) **Cross-modality expansion.** Integrating specialized operators for graph databases and vector stores to fully support the “compound AI” requirements of enterprise environments.

Acknowledgments

We would like to thank Pouya Pezeshkpour for his valuable feedback and review of this paper.

References

- [1] Alnur Ali, Ashutosh Baheti, Jonathan Chang, Ta-Chung Chi, Brandon Cui, Andrew Drozdov, Jonathan Frankle, Abhay Gupta, Pallavi Koppol, Sean Kulinski, Jonathan Li, Dipendra Misra, Krista Opsahl-Ong, Jose Javier Gonzalez Ortiz, Matei Zaharia, and Yue Zhang. 2025. A State-of-the-Art SQL Reasoning Model using RLVR. *CoRR* abs/2509.21459 (2025). arXiv:2509.21459 doi:10.48550/ARXIV.2509.21459
- [2] Jan-Micha Bodensohn, Ulf Brackmann, Liane Vogel, Anupam Sanghi, and Carsten Binnig. 2025. Unveiling Challenges for LLMs in Enterprise Data Engineering. *Proc. VLDB Endow.* 19, 2 (Oct. 2025), 196–209. doi:10.14778/3773749.3773758
- [3] Harrison Chase. 2022. LangChain. <https://github.com/langchain-ai/langchain>. GitHub repository.
- [4] Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. 2026. CRMarena-Pro: Holistic Assessment of LLM Agents Across Diverse Business Scenarios and Interactions. *Transactions on Machine Learning Research* (2026). <https://openreview.net/forum?id=EP1pe3Fx1x>
- [5] Eser Kandogan, Nikita Bhutani, Dan Zhang, Rafael Li Chen, Sairam Gurajada, and Esteveam Hruschka. 2025. Orchestrating Agents and Data for Enterprise: A Blueprint Architecture for Compound AI. In *41st IEEE International Conference*

- on Data Engineering, ICDE 2025 - Workshops, Hong Kong, May 19-23, 2025. IEEE, 18–27. doi:10.1109/ICDEW67478.2025.00007
- [6] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhaman A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *The Twelfth International Conference on Learning Representations*. <https://openreview.net/forum?id=sY5N0zY5Od>
 - [7] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. 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 *Proceedings of the 37th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1835, 28 pages.
 - [8] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. arXiv:2405.14696 [cs.CL] <https://arxiv.org/abs/2405.14696>
 - [9] Jerry Liu. 2022. *Llamaindex*. doi:10.5281/zenodo.1234
 - [10] Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. 2024. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate SQL queries from natural language prompts. https://huggingface.co/datasets/gretelai/synthetic_text_to_sql
 - [11] Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. 2025. From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. arXiv:2410.01066 [cs.CL] <https://arxiv.org/abs/2410.01066>
 - [12] Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A. Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, Shu Liu, Tianneng Shi, Xiaoyuan Liu, Jared Quincy Davis, Emmanuele Lacavalla, Alessandro Basile, Shuyi Yang, Paul Castro, Daniel Kang, Joseph E. Gonzalez, Koushik Sen, Dawn Song, Ion Stoica, Matei Zaharia, and Marquita Ellis. 2025. Measuring Agents in Production. *CoRR* abs/2512.04123 (2025). arXiv:2512.04123 doi:10.48550/ARXIV.2512.04123
 - [13] Liana Patel, Siddharth Jha, Melissa Z. Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2024. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. <https://api.semanticscholar.org/CorpusID:271218837>
 - [14] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. 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, 2609–2634. doi:10.18653/v1/2023.acl-long.147
 - [15] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.
 - [16] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=WE_vluYUL-X
 - [17] 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 *ICLR*. OpenReview.net. <http://dblp.uni-trier.de/db/conf/iclr/iclr2023.html#YaoZYDSN023>
 - [18] 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. doi:10.18653/v1/D18-1425

A Limitations.

Our current framework exhibits several limitations. First, tool interoperability remains a bottleneck; input/output mismatches between operators can cause planning failures that require manual schema tuning. Second, while the iterative planner offers superior robustness through feedback loops, it introduces significantly

higher latency compared to single-shot methods, which may be unsuitable for real-time applications. Third, the quality of generated plans remains highly dependent on the reasoning capabilities of the underlying LLM, which varies across model classes and prompt configurations. Finally, ensuring consistent data provenance across complex joins and multi-modal retrievals remains an ongoing challenge for complete auditability.

B Experiments

B.1 Setup

We implement our methods using the Blue Architecture framework for enterprise agentic workflows.² For benchmarking, we draw on the logic of standard text-to-SQL datasets (e.g., Spider [18]) but synthetically create a new dataset to reflect our open-world assumption. We use an LLM (Grok) to generate 66 questions of varying difficulty (Easy, Medium, Hard, Very Hard) over a schema involving PostgreSQL and web search tools.

Tool Set. Our experiments use a shared operator set:

- NL2SQL / SMARTNL2SQL: Convert natural language to SQL (running either on the database or on intermediate rows).
- NL2LLM / ROWWISE_NL2LLM: Open-domain knowledge queries via LLM tools.
- SELECT, JOIN, APPEND, COUNT: Standard relational-style data manipulation operators.
- JOIN_2: A simplified binary join operator introduced because a generic N-way JOIN proved too complex for reliable LLM use in our setting, consistent with prior reports on operator complexity in text-to-SQL agents [1].

Evaluation Protocol. Because enterprise settings rarely provide gold-standard QA datasets for open-ended retrieval, we adopt a *collective judging* methodology similar to compound-agent evaluation practices [10]. For each question, we run every planner 5 times, shuffle and anonymize all generated plans and answers, and ask an LLM-as-judge to identify correct outputs. This mitigates variance across runs and reduces bias toward specific planner identities, at the cost of relying on LLM-based evaluation.

B.2 Results and Discussion

Figure 6 summarizes our empirical comparison across planners, mapping their selection frequency by an LLM-as-judge against their average latency.

Performance by Planner Type.

- **Single-Shot Tree Planning (reasoning).** This pipeline relies on a state-of-the-art reasoning model (GPT-5 class) acting as a **monolithic planner**. It dominates the benchmark, achieving the highest number of top-tier evaluations (41.5 LLM Best Picks) and the fastest average latency (72.5 seconds). However, because this variant utilizes a significantly more powerful and expensive model than the others (which rely on standard GPT-4o-class models), it represents a ceiling of what is possible rather than a direct algorithmic baseline.
- **Single-Shot Tree Planning (guided NLMerge, inner → outer).** Utilizing a standard LLM, this guided tree variant

²<https://github.com/megagonlabs/blue>

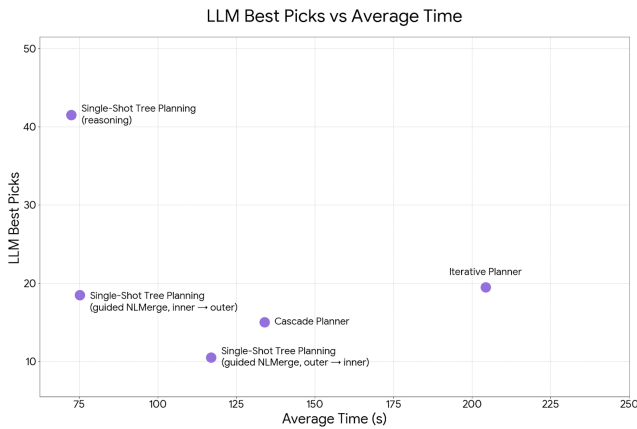


Figure 6: Planner performance as selected by an LLM-as-judge against average latency (lower is faster). Data represents averages across a 66-question dataset of varying difficulties.

is highly efficient. It yields the second-best latency (75.2 seconds) and a solid performance rate (18.5 picks), making it a strong baseline for standard complexity tasks.

- **Single-Shot Tree Planning (guided NLMerge, outer → inner).** Conversely, reversing the generation direction causes the standard LLM to struggle in comparison. It takes longer (117 seconds) and yields fewer optimal results (10.5 picks), highlighting the sensitivity of single-shot planning to prompt design and operator ordering.
- **Cascade Planner.** Despite its theoretical advantage of rapid execution initiation via pipelining, the Cascade Planner records moderate performance (15 picks) and relatively high average latency (134 seconds). We hypothesize that this latency

inflation occurs because, when the Cascade Planner fails, its refinement steps dynamically generate new iterations of the plan. The overhead of repeatedly evaluating and correcting these sub-plans heavily penalizes its average execution time.

- **Iterative Planner.** The Iterative Planner serves as a “pay latency for robustness” option. While it generates a respectable number of correct, self-corrected plans (19.5 picks), its stop-and-wait loop makes it the slowest method by far, averaging 204.3 seconds—nearly three times slower than the leading single-shot tree.

The Case for Dynamic Routing Across Difficulties.

At first glance, the overwhelming success of the *Single-Shot Tree Planning (reasoning)* variant might suggest that the other pipelines are obsolete. However, it is crucial to note that these metrics represent averages across 66 questions ranging from Easy to Very Hard.

The dominance of the GPT-5 class reasoning model masks the fact that for many Easy and Medium queries, the much lighter, cost-effective *Single-Shot Tree Planning (guided NLMerge, inner → outer)* variant achieves identical success rates. Furthermore, while the Cascade Planner’s refinement cycles slow its average time, its modularity remains highly beneficial for isolating hallucinations on multi-step schemas.

Consequently, a “one-size-fits-all” approach is suboptimal in a production environment. Instead, these results argue strongly that a planning strategy should be carefully selected for each individual question and its predicted difficulty. An ideal architecture would employ a dynamic meta-router that defaults to fast, guided NLMerge pipelines for straightforward queries, reserving the expensive GPT-5 reasoning model or the highly adaptive Iterative Planner strictly for Very Hard, ambiguous queries where self-correction and deep reasoning justify the cost and latency overhead.