LUMILAKE: An Agentic Analytics Engine for AI4Science

Zhengyuan Su, Noppanat Wadlom, Junyi Shen, Yicong Huang*, Wentao Wu+, Yao Lu
National University of Singapore, Databricks*, Microsoft Research+

Abstract

In the emerging *agentic analytics*, the fundamental unit of computation is no longer a SQL query but an *agentic workflow*: a DAG of agent operations that compose SQL queries, object fetches, LLM/VLM invocations, and multi-round subagent interactions. These workflows are computationally explosive—a single seven-node financial-analysis template applied to 160 stock symbols generates over 800 GPU-resident LLM/VLM calls with overlapping prefixes—yet existing systems treat each call in isolation. We present LUMILAKE, an agentic analytics engine that models workflows as first-class, introspectable DAGs and addresses three challenges: (i) LLM-aware query optimization that consolidates shared prefixes to maximize KV-cache reuse, (ii) a multi-tenant runtime that dispatches heterogeneous operators across mixed hardware with cross-tenant deduplication, and (iii) governance by design with semantic lineage at every agent step. Deployed at the National University of Singapore, LUMILAKE achieves up to 3.8× speedup over Ray on six real workflows from clinical decision support, materials spectroscopy, and financial analysis on commodity GPUs. Our engine is open-sourced at https://github.com/mlsys-io/lumilake.

Keywords

Agentic analytics; AI4Science workflows; LLM workflow optimization; Lakehouse analytics; Semantic provenance

1 Introduction

For decades, data analytics engines have been defined by the primitives they execute. SQL engines process declarative queries over relational tables; Spark [22] and its successors run distributed dataflow programs. The two paradigms shared storage, often via a data lakehouse [1], but were connected only through manual hand-offs or brittle ETL. In both cases, the unit of work that the engine optimizes, schedules, and governs was a human-authored query or script.

We argue that a paradigm shift is underway [8]. In the emerging *agentic analytics*, the fundamental unit of computation is an *agentic workflow*: a DAG whose nodes are *agent operations*, each of which spawns sub-operations such as data retrieval, model invocations, or subagent interactions. At the top level, a user submits a workflow DAG that orchestrates multiple agent operations; at the node level, each operation instantiates a sub-DAG of actions defined by a *template*—issuing SQL queries, fetching objects from storage, invoking models, launching subagents—with the agent generating the runtime parameters that bind each action to a specific input [18].

This compositional, two-level structure distinguishes agentic workflows from flat SQL query plans and static Spark dataflow graphs: the engine must optimize not only the top-level DAG but also the templated sub-operations within each node. Just as the transition from hand-written queries to cost-based optimizers transformed relational analytics, the transition from human-authored scripts to agent-authored pipelines demands a new engine design

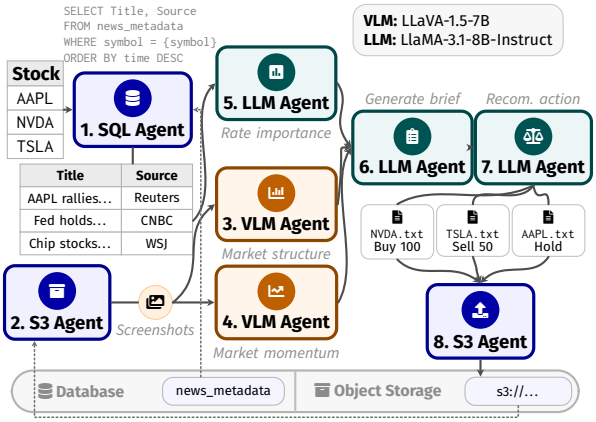


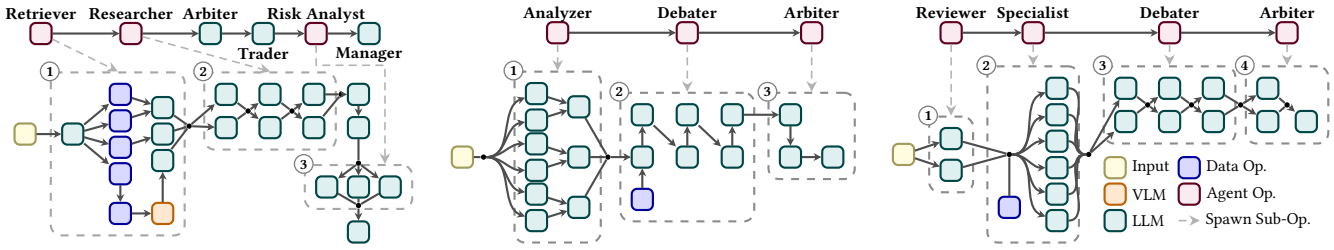
Figure 1: An agentic analytics workflow for financial analysis, which generates over 800 GPU-resident calls for 160 symbols.

that understands, optimizes, and governs agentic workflows as first-class objects. This shift is already visible in AI4Science [19, 23], as illustrated with workflows from real scientific and industrial settings in Section 2.

Consider the financial-analysis workflow in Figure 1: given a stock symbol, a top-level DAG of seven agent operations produces a Buy/Sell/Hold recommendation, chaining data retrieval (Nodes 1–2), parallel VLM analysis (Nodes 3–4), and sequential LLM reasoning (Nodes 5–7); an eighth node uploads the output to object storage. Each node follows its own template: Node 1 issues a SQL query over a relational database; Node 2 fetches market snapshots from object storage; Nodes 3–4 each invoke a VLM to analyze every retrieved image; Nodes 5–7 compose multi-step LLM calls for importance rating, synthesis, and recommendation. Running this workflow for 160 symbols instantiates the same templates 160 times, producing over 800 GPU-resident LLM/VLM calls, most sharing identical system prompts and overlapping prefixes. In a Ray [9]+vLLM [5] baseline, each call is served independently with no KV-cache sharing across nodes or symbols, and the job takes hours on a university cluster.

The above example exposes three challenges that existing systems leave unaddressed: (1) agentic workflows are *computationally explosive*, amplifying each input through two levels of template composition into hundreds or thousands of LLM/VLM calls, yet serving systems [5, 24] treat each call in isolation; (2) a single DAG mixes *heterogeneous operators*—SQL, object fetches, VLM, LLM, diffusion—on fragmented infrastructure where orchestration frameworks [4, 19] treat invocations as black-box UDFs; (3) tracing outputs requires *semantic provenance* across the two-level hierarchy, unlike Spark's fault-tolerance lineage [7, 11, 13].

Building on our prior work—Halo [14] (DAG-level batch optimization), Helium [17] (cache-aware LLM serving), and FlowMesh [15]



(a) Financial Analytics (Q2-Trading) (b) Materials Spectroscopy (Q6) (c) Clinical Decision Support (Q5)

Figure 2: Three representative agentic analytics workflows. Top: agent-level DAG; bottom: expanded sub-operations. Edges denote data dependencies.

(multi-tenant service fabric)—we introduce Lumilake, an analytics engine with agentic workflows as the primary compute primitive:

- *LLM-aware query optimization* that flattens template structure, consolidates shared prefixes, and schedules execution to maximize KV-cache reuse;
- *Multi-tenant runtime* that dispatches heterogeneous operators across mixed hardware with cross-tenant deduplication;
- *Governance by design* with semantic lineage at every agent step at under 2% overhead.

Deployed at the National University of Singapore, Lumilake achieves up to 3.8× speedup over Ray [9] across six workflows spanning three AI4Science domains [16].

2 Background

2.1 Agentic Analytics in AI4Science

Agentic workflows are replacing SQL queries and Python scripts as the primary analytical primitive. We describe six workflows drawn from real scientific and industrial settings that also serve as our evaluation benchmarks in Section 4. Figure 2 illustrates three representative workflows.

Financial data analytics. The following four workflows reflect common patterns in AI4Finance analysis. (1) `Q1-Social` models a text-only analyst roundtable: given a stock symbol, agents issue four SQL retrieval queries over financial, insider-trading, market, and news data, summarize the retrieved materials, and follow a three-round discussion among four analyst personas before a reporter synthesizes the report. (2) `Q2-Trading`, modeled after the TradingAgents framework [21], adds multimodal analysis: in addition to structured data retrieval, a VLM agent analyzes market snapshot images, and agents follow a multi-round debate template covering bull/bear researcher debate, risk analysis, trader decisions, and fund-manager synthesis, combining SQL, S3 retrieval, VLM, and LLM operators in a single 18-node DAG. (3) `Q3-ImageGen` captures a news-to-infographic generation scenario: given a stock symbol, the workflow retrieves related news, summarizes it, drafts competing render prompts via a critique template, invokes a diffusion model, critiques outputs with a VLM, and produces a final image artifact. (4) `Q4-Summary` represents an ETL-style ingestion pipeline commonly used by financial data vendors: it takes news images from object storage, analyzes each with a VLM, and runs three parallel drafting-and-critique branches following a reflection template before a summarizer writes the final output.

Table 1: Comparison of systems w.r.t. cross-query LLM-aware optimization (C1), KV-cache-aware scheduling (C2), semantic provenance (C3), and first-class support for agentic DAGs (C4). ✓ = supported, ○ = partial, × = not supported.

Category	System(s)	C1	C2	C3	C4
LLM Inference	vLLM [5], SGLang [24]	×	○	×	×
Orchestration	AutoGen [19], LangGraph [4]	×	×	×	○
Classic Data Sys.	Ray [9], Spark [22]	○	×	○	○
LLM Data Sys.	Palimpzest [7], LOTUS [11]	×	×	×	○
Workflow Sys.	AFlow [23], JellyBean [20]	×	×	×	✓
Ours	Lumilake	✓	✓	✓	✓

Clinical decision support. The `Q5-Healthcare` workflow is an osteoporosis decision-support pipeline developed with experts at the National University Hospital (NUH) Singapore [10]. A data-retrieval agent issues SQL over patient records; six specialist agents cross-reference each patient report against micro-review summaries from different clinical angles, each following a template with distinct clinical guidelines from practicing endocrinologists; a three-round pro/con debate template produces a final diagnostic report with contradiction checking. A single patient case requires nearly 20 LLM calls; screening 320 patients requires over 6,000 calls.

Materials spectroscopy. The `Q6-Materials` workflow is a spectroscopy analysis pipeline developed with researchers from the NUS Department of Chemical Engineering [3, 6]. Three expert branches (feature identification, quantitative extraction, physical interpretation) each follow a template with two analyst subagents using different guided prompts; a three-round cross-branch critique-and-rebuttal debate challenges the analyses before a summary chain synthesizes the final report. Analyzing 160 spectra produces over 1,200 LLM calls.

2.2 Prior Systems for Agentic Workflows

Table 1 summarizes the capabilities of existing systems along four axes that correspond to the challenges identified in Section 1: cross-query LLM-aware optimization (C1), KV-cache-aware scheduling (C2), semantic provenance (C3), and first-class support for agentic DAGs with templated sub-operations (C4). No prior system addresses all four. We discuss each category below.

LLM inference engines [5, 24] achieve high throughput via continuous batching and prefix caching but operate at request granularity:

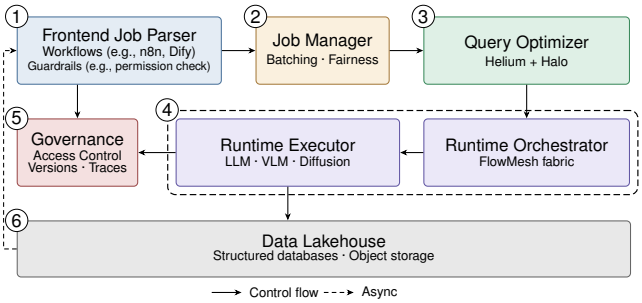


Figure 3: LUMILAKE system architecture.

they cannot detect shared prefixes *across* workflow instances or reorder execution across operators. Orchestration frameworks [4, 19] *define* templated agent patterns but do not *optimize across* instances; they treat every invocation as a black-box UDF with no cross-query optimization, KV-cache visibility, or built-in provenance. Classic data systems [9, 22] provide resource management but lack LLM-semantic awareness to deduplicate across tenants. Semantic data systems [7, 11, 13] target flat operator pipelines rather than compositional two-level DAGs. Workflow systems [20, 23] improve the *logical* plan but delegate execution to black-box backends; LUMILAKE is complementary, optimizing the *physical* execution.

Our prior work. Helium [17] optimizes batched execution of agentic workflows via greedy DFS-based cache-aware scheduling, which scales linearly but assumes all nodes share one base LLM. Halo [14] uses epoch-based DP to support heterogeneous operators, but it scales exponentially to the workflow size. FlowMesh [15] proposes a multi-tenant service fabric that deduplicates tasks across concurrent users. LUMILAKE unifies all three into a single multi-tenant, multi-template agentic analytics engine.

3 System Overview

As shown in Figure 3, LUMILAKE is built on a data lakehouse [1] and comprises six components: a *workflow parser* that validates incoming jobs against guardrails; a *job manager* that constructs and dispatches batches; a *query optimizer* that rewrites and schedules batched workflow DAGs; a *runtime processor* (FlowMesh [15]); a *governance layer* for semantic lineage; and a *storage layer*.

Job manager. Upon ingestion, a job's input is split along the input dimension (e.g., 160 stock symbols become 160 slices), yielding uniform-cost scheduling units that prevent head-of-line blocking. Pending slices are maintained in per-priority, per-user queues with round-robin to prevent monopolization; a starvation counter pins long-waiting jobs into the next batch. From the candidate pool, the manager selects a batch of size B that maximizes intra-batch affinity, i.e., Jaccard similarity over model identifiers and system-prompt sets, via agglomerative hierarchical clustering, which promotes resource reuse within the batch.

Query optimizer. The selected batch is sent to the query optimizer, which (1) rewrites the batch of workflow graphs into a single optimized graph and (2) generates an execution schedule maximizing resource reuse.

Graph rewriting aggregates slices from the same job, then applies three semantics-preserving rewrites: (1) *VLM splitting* decomposes each VLM node into an embedding and a reasoning sub-node, exposing embeddings for deduplication when multiple agents analyze the same image; (2) *Node absorption* merges lightweight formatting nodes into their GPU-bound successors; (3) *Subgraph deduplication* computes structural signatures over operator specifications and parent signatures, merging identical nodes within and across jobs.

Hybrid scheduling. The rewritten graph may contain 30–50 nodes with heterogeneous operator types. Helium [17] scales linearly via greedy DFS over a templated radix tree but assumes all nodes share one base LLM. Halo [14] co-schedules heterogeneous CPU/GPU operators via epoch-based dynamic programming but is exponential in graph size and struggles beyond ~20 nodes. Both systems pair one workflow template with one user's input batch. Our key observation is that multimodal agentic workflows still contain large homogeneous subgraphs (all text-generation nodes sharing one LLM), with heterogeneous operators appearing mainly at DAG boundaries. Based on the observation, we propose a hybrid algorithm that gets the best of both. We identify connected components of same-model, text-only LLM nodes and *contract* each into a super-node, applying Helium's cache-aware greedy DFS within each component to optimize KV-cache reuse order. We then schedule the contracted graph (typically 5–15 nodes) via Halo's epoch-based DP solver to optimize cross-operator placement and model switching. This hybrid approach keeps optimizer overhead at ~1.4 s per batch while scaling to 50+ node DAGs, on which Halo alone would time out.

Cycle-safe contraction. Naïve contraction can introduce cycles. Consider three LLM nodes A, B, C sharing the same base LLM and a VLM node D, with edges A→B, B→C, A→D, and D→C. Contracting A, B, C into a super-node T creates a cycle T→D→T, violating the DAG property. Our algorithm checks cycle safety before contracting: contraction is valid only if no external node has both an incoming and outgoing dependency on the component. Any violating component is split at the ancestor boundary and re-partitioned, guaranteeing the contracted graph remains a valid DAG.

Multi-tenant runtime. The runtime is built on FlowMesh [15], extended with heterogeneous operator support and scheduling hints. LUMILAKE passes the optimizer's output as two hints: a *hard* task-to-worker assignment (pinning each task to a specific GPU or CPU worker) and a *soft* per-worker execution order (used as a tiebreaker among ready tasks). The soft constraint allows FlowMesh to merge tasks destined for different execution slots when doing so increases the inference batch size, improving per-node throughput. The executor pool supports SQL and S3 data retrieval, image embedding, VLM understanding, LLM text generation, and diffusion-based image generation. In multi-tenant settings, researchers on the same cluster share base LLMs and system-prompt fragments, creating cross-job prefix and weight reuse invisible to single-user systems.

Governance layer. LUMILAKE records provenance as a by-product of execution: every executor logs timestamped events (model initialization, generation start/end, intermediate outputs) into an in-memory DuckDB instance checkpointed to the lakehouse. This provides both semantic auditing (which inputs, prompts, and model versions contributed to a result) and performance diagnosis. The combined overhead of query optimization and governance tracking is below 2% of end-to-end time across all workloads.

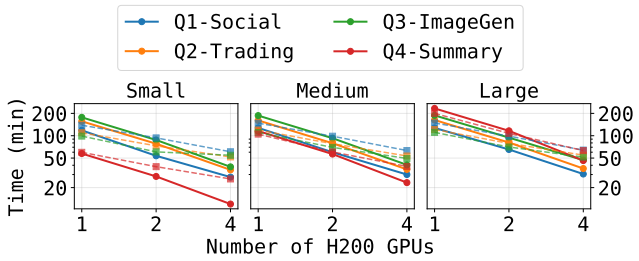


Figure 4: End-to-end time for AI4Finance workloads across data scales and GPU counts. Solid: LUMILAKE; dashed: Ray.

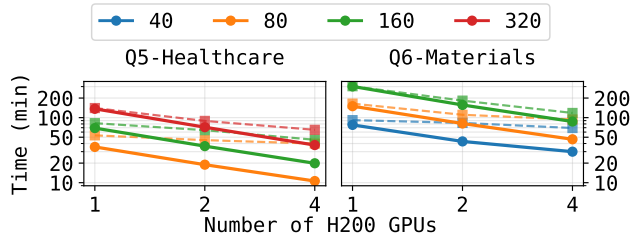


Figure 5: End-to-end time for AI4Science workloads across data scales and H200 GPU counts.

4 Evaluation

Experiment setup. LUMILAKE is deployed at the National University of Singapore in collaboration with domain experts in clinical medicine and materials science. The hardware is representative of a university research group: four local NVIDIA H200 GPUs, with up to eight on-demand NVIDIA RTX 5090 GPUs from a cloud marketplace. We evaluate on the six workflows `Q1`–`Q6` described in Section 2.1. For AI4Finance workloads, we prepare three data splits of 1 GB (small), 10 GB (medium), and 100 GB (large); structured tables occupy one-third of each split, with the remainder consisting of raw HTML files and market snapshot images. For healthcare, we collect reports for 320 patients from the NHANES 2017–2020 Pre-pandemic Data Files [10]. For materials science, we use up to 160 variant spectra from [6]. We compare against Ray [9] with vLLM [5] as the inference backend—the de facto standard for deploying ML workloads on commodity clusters. Request-level prefix caching (e.g., SGLang [24]) is orthogonal to and composable with LUMILAKE's workflow-level optimization across operators and tenants.

End-to-end performance (Figure 4, Figure 5). With a single GPU, Ray is competitive on some workloads because continuous batching via vLLM yields a larger effective batch size. However, as GPUs increase, LUMILAKE's schedule-driven batching enables better reuse of model weights and KV-cache prefixes across nodes; with four GPUs, LUMILAKE outperforms Ray on all six workloads, averaging 1.7× speedup (up to 2.2×) on financial workloads and 2.2× (up to 3.8×) on AI4Science workloads. LUMILAKE scales near-linearly with GPU count; with four H200s, it diagnoses over 300 patients or analyzes 100 spectra within one hour.

Cross-tenant merging. When two researchers submit overlapping workflows concurrently—`Q1-Social` and a text-only variant of `Q2-Trading` over the same 160 stock symbols—LUMILAKE deduplicates shared retrieval and analysis stages across the two workflows, completing both in 92.3 minutes versus 126.5 minutes when they are processed sequentially. This 1.37× speedup comes from cross-tenant deduplication, confirming that multi-tenancy creates substantial optimization opportunities invisible to per-user systems.

Priority scheduling. When a medium-priority `Q2-Trading` job and a low-priority `Q4-Summary` job are submitted concurrently, both jobs make progress from the start, but the medium-priority job advances significantly faster. Once it completes, the low-priority job accelerates to finish, confirming that LUMILAKE respects priorities, avoids starvation, and elastically reallocates freed resources.

5 Discussion and Conclusion

What makes agentic analytics different? A key lesson from our experiments is that the optimization surface of agentic workflows is unlike that of SQL or Spark: the bottleneck is not data shuffling or I/O, but the sheer volume of LLM calls and model switches, and the primary lever is exploiting the structural regularity that templates create across inputs and users. This suggests that future agentic engines should be co-designed with the serving layer, treating KV-cache state, model residency, and prompt structure as first-class objects in the optimizer's cost model, rather than layered on top of general-purpose infrastructure.

AI4Science, auto research, and agentic analytics. Scientific research is inherently analytics-heavy: querying databases, cross-referencing multi-modal evidence, and iterating through expert review; agentic workflows automate exactly these central activities [2]. The emerging Auto Research paradigm [12], in which agents autonomously formulate hypotheses, run experiments, and synthesize findings, amplifies this by orders of magnitude: a single research campaign may launch thousands of agentic analytics runs. Efficient, governed execution of such workloads is precisely the problem LUMILAKE addresses.

Scope and limitations. LUMILAKE targets *template-based* agentic workflows whose top-level DAG and per-node sub-DAGs are fully specified before execution—the dominant pattern in AI4Science, where workflows are authored once by domain experts and instantiated over many inputs. Dynamically generated DAGs, in which agents decide at runtime which sub-operations to spawn (e.g., adaptive tool-use agents), would require online re-optimization, which we leave to future work.

Conclusion. We present LUMILAKE, an agentic analytics engine that treats template-based workflows as first-class DAGs over a data lakehouse. By combining LLM-aware query optimization, multi-tenant scheduling, and built-in governance, LUMILAKE outperforms Ray across all workloads on multi-GPU setups, scales to 50+ node DAGs with under 2% overhead, and achieves 1.37× speedup from cross-tenant deduplication—demonstrating that database principles can tame the computational explosion of agentic analytics.

Acknowledgments. We thank our domain collaborators: clinicians at the National University Hospital (Singapore) for the osteoporosis decision-support workflow, and researchers at the NUS Department of Chemical Engineering for the materials spectroscopy pipeline.

References

[1] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. Online.

[2] Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous Chemical Research with Large Language Models. *Nature* 624, 7992 (2023), 570–578.

[3] Alireza Ghafarollahi and Markus J. Buehler. 2025. SciAgents: Automating Scientific Discovery through Bioinspired Multi-Agent Intelligent Graph Reasoning. *Advanced Materials* 37, 22 (2025), 2413523.

[4] LangChain Inc. 2024. LangGraph: Low-level Orchestration Framework for Building Stateful Agents. https://github.com/langchain-ai/langgraph

[5] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. ACM, 611–626.

[6] Nengxu Li, Xiuxiu Niu, Zijing Dong, Jingcong Hu, Ran Luo, Shuihua Yang, Qilin Zhou, Zhuojie Shi, Jinxi Chen, Xinyi Du, et al. 2025. Optimal perovskite vapor partitioning on textured silicon for high-stability tandem solar cells. *Science* 390, 6779 (2025), eadz3698.

[7] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[8] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, et al. 2026. Supporting Our AI Overlords: Redesigning Data Systems to Be Agent-First. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. Santa Cruz, CA, USA.

[9] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 561–577.

[10] National Center for Health Statistics (NCHS). 2022. *National Health and Nutrition Examination Survey (NHANES) 2017–March 2020 Prepandemic File*. Technical Report. Centers for Disease Control and Prevention. https://wwwn.cdc.gov/nchs/nhanes/analyticguidelines.aspx

[11] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4171–4184.

[12] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. Agent Laboratory: Using LLM Agents as Research Assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. 5977–6043.

[13] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. *Proceedings of the VLDB Endowment* 18, 9 (2025), 3035–3048.

[14] Junyi Shen, Noppanat Wadlom, and Yao Lu. 2025. Batch Query Processing and Optimization for Agentic Workflows. *arXiv preprint arXiv:2509.02121* (2025).

[15] Junyi Shen, Noppanat Wadlom, Lingfeng Zhou, Dequan Wang, Xu Miao, Lei Fang, and Yao Lu. 2025. FlowMesh: A Service Fabric for Composable LLM Workflows. *arXiv preprint arXiv:2510.26913* (2025).

[16] Zhengyuan Su, Noppanat Wadlom, Junyi Shen, Peisong Zhang, Ran Luo, Dianbo Liu, Yi Hou, Yicong Huang, Wentao Wu, and Yao Lu. 2026. Lumilake: An Agentic Analytics Engine for AI4Science (Extended). https://lumilake.short.gy/lumilake-extended

[17] Noppanat Wadlom, Junyi Shen, and Yao Lu. 2026. Efficient LLM Serving for Agentic Workflows: A Data Systems Perspective. *arXiv preprint arXiv:2603.16104* (2026).

[18] Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. 2023. Scientific Discovery in the Age of Artificial Intelligence. *Nature* 620, 7972 (2023), 47–60.

[19] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2024. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. In *Proceedings of the Conference on Language Modeling (COLM)*.

[20] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures. *Proceedings of the VLDB Endowment* 16, 3 (2022), 406–419.

[21] Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. 2025. TradingAgents: Multi-Agents LLM Financial Trading Framework. arXiv:2412.20138 [q-fin.TR] https://arxiv.org/abs/2412.20138

[22] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 15–28.

[23] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. 2025. AFlow: Automating Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

[24] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 37. 62557–62583.